



TESIS - TE142599

EVOLUSI DINAMIS PERILAKU NON-PLAYER CHARACTER PADA GAME SPACE SHOOTER MENGUNAKAN NSGA-II

DARMAWAN ADITAMA
2214205013

DOSEN PEMBIMBING
Dr. Supeno Mardi Susiki Nugroho, S.T., MT.
Mochamad Hariadi, ST., M.Sc., Ph.D.

PROGRAM MAGISTER
BIDANG KEAHLIAN JARINGAN CERDAS MULTIMEDIA
JURUSAN TEKNIK ELEKTRO
FAKULTAS TEKNOLOGI INDUSTRI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2017



TESIS - TE142599

DYNAMIC EVOLUTION BEHAVIOR FOR NON-PLAYER CHARACTER ON SPACE SHOOTER GAME USING NSGA-II

DARMAWAN ADITAMA
2214205013

DOSEN PEMBIMBING
Dr. Supeno Mardi Susiki Nugroho, S.T., MT.
Mochamad Hariadi, ST., M.Sc., Ph.D.

PROGRAM MAGISTER
BIDANG KEAHLIAN JARINGAN CERDAS MULTIMEDIA
JURUSAN TEKNIK ELEKTRO
FAKULTAS TEKNOLOGI INDUSTRI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2017

LEMBAR PENGESAHAN

Tesis disusun untuk memenuhi salah satu syarat memperoleh gelar
Magister Teknik (M.T)
di
Institut Teknologi Sepuluh Nopember

oleh:

DARMAWAN ADITAMA
NRP. 2214205013

Tanggal Ujian : 09 Januari 2017
Periode Wisuda: 115 - (Maret 2017)

Disetujui oleh:

1. Dr. Supeno Mardj Susiki Nugroho, S.T., MT. (Pembimbing I)
NIP: 197003131995121001

2. Mochamad Hariadi, ST., M.Sc., Ph.D. (Pembimbing II)
NIP: 196912091997031002

3. Dr. Surya Sumpeno, ST., M.Sc. (Penguji)
NIP: 196906131997021003

4. Dr. Diah Puspito Wulandari, ST., M.Sc. (Penguji)
NIP: 198012192005012001



Direktur Program Pascasarjana

Prof. Ir. Djauhar Manfaat, M.Sc, Ph.D
NIP. 19601202 198701 1 001

PERNYATAAN KEASLIAN TESIS

Dengan ini saya menyatakan bahwa isi keseluruhan Tesis saya dengan judul “**EVOLUSI DINAMIS PERILAKU *NON-PLAYER CHARACTER* PADA *GAME SPACE SHOOTER* MENGGUNAKAN *NSGA-II***” adalah benar-benar hasil karya intelektual mandiri, diselesaikan tanpa menggunakan bahan-bahan yang tidak diijinkan dan bukan merupakan karya pihak lain yang saya akui sebagai karya sendiri.

Semua referensi yang dikutip maupun dirujuk telah ditulis secara lengkap pada daftar pustaka. Apabila ternyata pernyataan ini tidak benar, saya bersedia menerima sanksi sesuai peraturan yang berlaku.

Surabaya, 15 Januari 2017

DARMAWAN ADITAMA
NRP. 2214205013

Halaman ini sengaja dikosongkan

EVOLUSI DINAMIS PERILAKU *NON-PLAYER CHARACTER* PADA *GAME SPACE SHOOTER* MENGGUNAKAN NSGA-II

Nama Mahasiswa : Darmawan Aditama

NRP : 2214205013

Dosen Pembimbing : 1. Dr. Supeno Mardi Susiki Nugroho, ST., MT.
2. Mochamad Hariadi, ST., M.Sc., Ph.D.

ABSTRAK

Dalam permainan *space shooter* terdapat musuh yang dikendalikan oleh *Non-Player Character* (NPC) statis dimana *player* dapat terus berusaha beradaptasi terhadap perilaku dari musuh. Pada akhirnya, permainan menjadi membosankan karena perilaku NPC telah diketahui oleh *player*. Dalam penelitian ini dikembangkan implementasi kecerdasan buatan dimana NPC dapat mengevolusi dirinya sendiri, sehingga mampu merespon perilaku *player*. Evolusi dinamis NPC dapat memberikan pengalaman bermain yang menyenangkan karena perilaku NPC dapat menyesuaikan perilaku *player*. *Non-Dominated Sorting Genetic Algorithm II* (NSGA-II) dalam simulasi digunakan untuk mengatur “titik evolusi” terhadap NPC. Dimana dengan memeriksa parameter keputusan dari perilaku *player* untuk dapat mengevolusi perilaku dari musuh (NPC). Kemudian *Non-Dominated Sorting Genetic Algorithm II* (NSGA-II) digunakan untuk mencari *optimal solution* untuk mengoptimalkan evolusi dinamis NPC pada level berikutnya. Hal ini bertujuan untuk menyeimbangkan perilaku musuh terhadap *player*. Simulasi menggunakan NSGA-II menghasilkan solusi optimal dalam bentuk grafik dengan 2 obyektif (*speed* dan *health*) dari NPC. Berdasarkan hasil simulasi, kestabilan berada pada generasi ke-5 dengan populasi=50, probabilitas persilangan (p_c) = 1, probabilitas mutasi (p_m)= $1/n$, indeks distribusi persilangan (η_c)=20, dan indeks distribusi mutasi (η_m)=20.

Kata Kunci: Algoritma Genetika, *Artificial Intelligent*, *Weapon Evolusi*,

halaman ini sengaja dikosongkan

DYNAMIC EVOLUTION BEHAVIOR FOR NON-PLAYER CHARACTER ON SPACE SHOOTER GAME USING NSGA-II

By : Darmawan Aditama
Student Identity Number : 2214205013
Supervisor(s) : 1. Dr. Supeno Mardi Susiki Nugroho, ST., MT.
2. Mochamad Hariadi, ST., M.Sc., Ph.D.

ABSTRACT

Abstrak – In Space Game, the NPC enemies have a static ability or behaviour where the player can be easily learn to adapted and challenge the NPCs. In the end, it makes the player become boring and don't want to play the game anymore. This research is develop an intelligent NPC that can be adapt to player style in playing the game. The NPC can be evolve and adjust itself based on the player behaviour in playing game and giving a proper difficulty to the game itself. This implementation will be make game more fun and enjoyable. NSGA II is the algorithm that will used to arrange the evolution of NPCs. This algorithm is used to evaluate the parameter that will determine NPCs behaviour change. Beside that, this research also used NSGA-II to optimized the paramater before it used to determined the NPCs behaviour change in every level of the game. Simulation it self used NSGA-II to produce optimal solution in graphic representation based on two objective parameter, speed and *health*. Result of simulation shown the optimized result will be produce the best solution after the five generation with total population is 50. crossover probability (p_c)=0.9, mutation probability (p_m)=1/n, index of distribution crossover (η_c)=20, index of distribution mutation (η_m)=20.

Keywords: *Genetics Algorithm, Artificial Intelligent, Damage Evolution*

Halaman ini sengaja dikosongkan

KATA PENGANTAR

Segala puji syukur kepada ALLAH SWT Tuhan Semesta Alam. Berkat rahmat-NYA, penulis dapat menyelesaikan tesis ini dengan baik. Tesis dengan judul “EVOLUSI DINAMIS PERILAKU *NON-PLAYER CHARACTER* DALAM PERMAINAN *SPACE SHOOTER* MENGGUNAKAN NSGA II” diselesaikan penulis dalam dua semester, yakni pada semester 4-5 program Pasca Sarjana ini. Tesis ini disusun guna memenuhi persyaratan untuk memperoleh gelar Magister Teknik pada bidang konsentrasi Teknologi Permainan, pada studi Jaringan Cerdas Multimedia, jurusan Teknik Elektro, Institut Teknologi Sepuluh Nopember Surabaya.

Keterbatasan kemampuan penulis dalam mengerjakan Tesis ini tidak terlalu menghambat penyelesaian penelitian karena begitu banyak perhatian dan bantuan dari rekan-rekan, para dosen, dan kerabat yang dengan ikhlas meluangkan waktu dan pikirannya untuk membantu penulis. Beberapa pihak yang penulis sebutkan berperan besar dalam penyusunan Tesis ini. Terima kasih penulis ucapkan terutama untuk:

1. M. Afnan dan Nur Latifah, Bapak dan Ibu penulis, serta adik – adik penulis yang telah memberikan dukungan moral dan material tanpa henti untuk melalui masa-masa perkuliahan program Magister. Dukungan mereka yang memberikan kekuatan penulis untuk melalui masa-masa sulit bahkan keruntuhan moril penulis.
2. Dr. Supeno Mardi Susiki Nugroho, S.T., M.T., sebagai Pembimbing I yang dengan sangat sabar membimbing penulis yang terus menerus melakukan kesalahan baik dalam penulisan maupun simulasi penelitian. Ucapan terima kasih kepada beliau bahkan terlalu kecil sebagai ungkapan rasa syukur atas kebaikan beliau dalam membimbing penulis selama dua semester pengerjaan penelitian.
3. Mochamad Hariadi, ST., M.Sc., Ph.D. sebagai Pembimbing II yang dengan sangat sabar membimbing penulis yang selalu tidak bisa mengutarakan maksud tesisnya sehingga sering gagal paham atas tujuan pengerjaan tesis

tersebut. Terima kasih telah menunjukkan kesalahan jalur yang telah dilalui penulis untuk menyelesaikan penelitian.

4. Dosen Pengajar Jurusan Teknik Elektro, khususnya bidang keahlian Jaringan Cerdas Multimedia yang telah memberikan pengetahuan dan pengalaman baru kepada penulis.
5. Sahabat terbaik, kawan seperjuangan game technology 2014 yang meskipun entah berada dimana kalian saat ini. Tidak akan pernah sekalipun penulis melupakan kebersamaan yang telah dilalui ketika menjalankan studi S2 di bidang jaringan cerdas multimedia. “kalian adalah cerita yang pantas untuk selalu diingat dimasa tua kelak”

Penulis sepenuhnya menyadari bahwa hasil karya ini sangatlah jauh dari kata “sempurna”. Walaupun penulis menganggapnya sebagai pencapaian yang tak disangka tapi tentulah masih banyak kekurangan yang dapat dikoreksi oleh pihak lain. Meskipun bagi penulis, ada rasa takjub dan bahagia yang mendalam karena telah menyelesaikannya. Dan semoga dapat dijadikan tema penelitian selanjutnya untuk menyempurnakan penelitian yang telah dikerjakan oleh penulis.

Surabaya, 15 Januari 2017

Penulis

DARMAWAN ADITAMA

DAFTAR ISI

LEMBAR PENGESAHAN	iii
PERNYATAAN KEASLIAN TESIS	v
ABSTRAK	vii
ABSTRACT	ix
KATA PENGANTAR	xi
DAFTAR ISI	xiii
DAFTAR GAMBAR	xvii
DAFTAR TABEL	xix
BAB I PENDAHULUAN	1
1.1. Latar Belakang	1
1.2. Rumusan Masalah	6
1.3. Batasan masalah	6
1.4. Tujuan Penelitian	6
1.5. Manfaat Penelitian	6
1.6. Kontribusi Penelitian	7
1.7. Sistematika Penulisan	7
BAB II KAJIAN PUSTAKA	9
2.1 Kajian Penelitian Terkait	9
2.1.1 Csikszentmihalyi's Flow Theory	9
2.1.2 Evolving Multimodal Network for Multitask <i>Games</i>	12
2.1.3 Evolusi Penyerangan NPC	16
2.1.4 Perilaku NPC Saat Pertahanan Tempur	17
2.2 Dasar Teori	17
2.2.1 Space Shooter	17
2.2.2 <i>Game Play</i> Pada Permainan	18
2.2.3 Perilaku	24
2.2.3.1 Adaptasi	24
2.2.3.2 Adaptasi Perilaku Evolusi dinamis	25

2.2.4 Multi Objective Optimization	26
2.2.5 Pareto Optimal	28
2.2.6 Algoritma <i>Genetika</i>	30
2.2.6.1 Pengertian Algoritma <i>Genetika</i>	30
2.2.6.2 Struktur Umum Algoritma <i>Genetika</i>	31
2.2.6.3 Pengkodean	33
2.2.6.3.1 Pengkodean Biner	33
2.2.6.3.2 Pengkodean Bilangan Riel	33
2.2.6.3.3 Pengkodean Nilai	34
2.2.6.4 Fungsi Evaluasi Kebugaran (Fitness)	34
2.2.6.5 Operator Genetik	34
2.2.6.5.1 Seleksi	35
2.2.6.5.2 Crossover	36
2.2.6.5.3 Mutasi	37
2.2.6.5.4 Elitism	38
2.2.6.5.5 Parameter Algoritma <i>Genetika</i>	38
2.2.7 Non-dominated Sorting Genetic Algorithm (NSGA-II)	41
2.2.7.1 Inisialisasi Populasi	43
2.2.7.2 Non-Dominated Sort	43
2.2.7.3 Crowding Distance	44
2.2.7.4 Selection	45
2.2.7.5 <i>Genetic</i> Operator	45
2.2.7.6 Recombination dan Seleksi	45
BAB III METODOLOGI PENELITIAN	47
3.1 Lingkup Penelitian	47
3.2 Test-bed Penelitian	48
3.2.1 Deskripsi <i>Game</i>	48
3.2.2 Desain 3D <i>Game</i>	49
3.2.3 Desain Senjata pada NPC	52
3.3 Tahapan Pembuatan Sistem	54
3.3.1 Evolusi Dinamis Dalam Permainan	55
3.3.2 Karakteristik <i>Player</i>	55

3.3.3 Perilaku NPC Terhadap <i>Player</i>	57
3.3.4 Spawn Awal (Populasi Awal).....	60
3.3.5 Dynamic Difficulty Adjustment (DDA)	60
3.3.6 NSGA II.....	61
3.3.6.1 Evaluasi Fungsi Objektif	63
3.3.6.2 Fungsi Obyektif	65
3.3.6.3 Pengurutan Nondominasi.....	68
3.3.6.4 Crowding Distance.....	69
3.3.6.5 Seleksi	69
3.3.6.6 Operator <i>Genetika</i>	70
3.4 Skenario Simulasi Dalam Permainan	71
3.5 Skenario Pengujian	73
3.5.1 Skenario Pengujian Pada Permainan	73
3.5.2 Skenario Simulasi menggunakan NSGA II.....	74
 BAB IV HASIL DAN PEMBAHASAN	 77
4.1 Simulasi Tanpa DDA dan Dengan DDA	77
4.1.1 Simulasi Permainan Tanpa DDA (Dinamic Difficulty Adjustment) dan Evolusi Dinamis NPC.....	78
4.1.2 Simulasi Permainan Dengan DDA (Dinamic Difficulty Adjustment).....	79
4.2 Simulasi Pada Permainan	81
4.2.1 Simulasi Permainan pada <i>Player</i> Pemula.....	82
4.2.2 Simulasi Permainan pada <i>Player</i> Handal	83
4.2.3 Grafik Perbandingan Evolusi Dinamis	85
4.3 Simulasi NSGA-II.....	86
4.3.1 Pengujian Performa NSGA-II.....	86
4.3.1.1 Pengujian NSGA II Pada Evolusi dinamis NPC.....	87
 BAB V KESIMPULAN.....	 93
5.1 Kesimpulan	93
5.2 Saran.....	93
 DAFTAR PUSTAKA	 95

BIODATA PENULIS.....	99
----------------------	----

DAFTAR GAMBAR

Gambar 1.1. <i>Game</i> Bergenre First Person Shooter (FPS).....	1
Gambar 1.2. <i>Game</i> Bergenre Third Person Shooter (TPS).....	2
Gambar 1.3 (a) & (b) <i>Game Space Shooter</i>	3
Gambar 2.1 Menunjukkan digram <i>flow</i>	11
Gambar 2.2 Jaringan untuk memainkan <i>game</i> multitask	14
Gambar 2.3 <i>Game Front/Back Ramming</i>	15
Gambar 2.4 <i>Game Predator/Prey</i>	16
Gambar 2.5 Contoh <i>game Space Shooter</i>	18
Gambar 2.6 Item Dengan Efek Lightning.....	19
Gambar 2.7. Ruang Keputusan dan Obyektif	27
Gambar 2.8 Konsep Pareto Optimal	29
Gambar 2.9 Blok Diagram Algoritma <i>Genetika</i>	31
Gambar 2.10 Ilustrasi Operator Dengan Satu Titik Persilangan.....	37
Gambar 2.11 Ilustrasi Operator Dengan Dua Titik Persilangan	37
Gambar 2.12 Ruang Keputusan dan Obyektif	42
Gambar 2.13 Blok Diagram NSGA-II	43
Gambar 3.1. Diagram Venn Lingkup Penelitian.....	48
Gambar 3.2 Desain 3D Pesawat Tempur	50
Gambar 3.3 Desain Item Dengan Efek <i>Lightning</i>	51
Gambar 3.4. Kondisi Saat <i>Player</i> Mendapatkan <i>Item Shield</i>	52
Gambar 3.5. Diagram Permainan.....	54
Gambar 3.6. Diagram Evolusi Dalam Permainan.....	55
Gambar 3.7. Diagram Karakteristik Player Dalam Permainan	57
Gambar 3.8. Evolusi Dinamis NPC	58
Gambar 3.9. Diagram Evolusi Dinamis NPC Dalam Permainan.....	59
Gambar 3.10. <i>Flow Channel</i> Dalam Skenario <i>Game</i>	60
Gambar 3.11 Multiobyektif di dalam <i>Flow Channel</i>	62
Gambar 3.12 Konsep Perilaku didalam <i>Flow Channel</i>	62
Gambar 3.13. Blok Diagram <i>Game</i> dengan DDA dan NSGA II.....	72
Gambar 4.1. Simulasi Permainan Tanpa DDA	78

Gambar 4.2 Kondisi User Interface Ketika <i>Player</i> Kalah	79
Gambar 4.3 Kondisi User Interface Ketika <i>Player</i> Menang	79
Gambar 4.4 Simulasi Permainan Pada Status TBD.....	80
Gambar 4.5 Menunjukkan Perubahan Level Setelah Satu Kali Level	80
Gambar 4.6 Menunjukkan Perubahan Level Setelah Satu Kali Level	81
Gambar 4.7 Simulasi Uji Permainan	82
Gambar 4.8 Grafik Perbandingan Evolusi <i>Speed</i> NPC	85
Gambar 4.9 Grafik Perbandingan Evolusi <i>Health</i> NPC	86
Gambar 4.10. Experiment dengan 50 populasi dan 100 Generasi.....	90

DAFTAR TABEL

Tabel 2.1 Jenis – Jenis Senjata dan Item <i>player</i>	20
Tabel 2.2 Spesifikasi Pesawat <i>Player</i>	21
Tabel 2.3 Macam-macam Senjata NPC	22
Tabel 2.4 Spesifikasi Pesawat NPC	23
Tabel 2.5 Contoh Pengkodean Biner	33
Tabel 2.6 Contoh Pengkodean Bilangan Riil atau Bilangan Permutasi.....	33
Tabel 2.7 Contoh Pengkodean Nilai	34
Tabel 2.8 Tabel Crossover 1-Titik	36
Tabel 2.9 Tabel Crossover 2-Titik	37
Tabel 2.10 Tabel Crossover Seragam	37
Tabel 3.1 Jenis – Jenis Senjata dan <i>Armor</i> Dalam Permainan.....	53
Tabel 3.2 Karakteristik <i>Player</i> Pemula.....	56
Tabel 3.3 Karakteristik <i>Player</i> Handal	56
Tabel 4.1 Data Pemain Pemula Dalam Satu <i>Game</i>	83
Tabel 4.2 Hasil Evolusi Dinamis NPC Pada Parameter Pemain Pemula.....	83
Tabel 4.3 Data Pemain Handal Dalam Satu <i>Game</i>	84
Tabel 4.4 Hasil Evolusi Dinamis NPC Pada Parameter Pemain Handal	84
Tabel 4.5 Konfigurasi Simulasi NSGA-II.....	87
Tabel 4.6 Menunjukkan Data Yang Digunakan Dalam Simulasi NSGA II	88
Tabel 4.7 Tabel Evolusi Dinamis NPC Dengan 50 Populasi dan 100 Generasi....	89

Halaman ini sengaja dikosongkan

BAB I

PENDAHULUAN

1.1 Latar Belakang

Teori *game* merupakan sebuah sistem matematika yang digunakan untuk menganalisa dan memprediksi bagaimana manusia berperilaku dalam situasi yang strategis ketika sedang memainkan *video game*. Analisa standart dalam menentukan keseimbangan permainan adalah dengan menganggap semua *player* membuat sebuah kepercayaan berdasarkan analisa dari perilaku dan kebiasaan musuh yang sedang dihadapi. Lantas memilih respon terbaik dalam memberikan kepercayaan (optimisasi) untuk menentukan perilakunya. Selanjutnya mengatur respon terbaik dari kepercayaan yang didapatkan untuk dapat menemukan perilaku yang konsisten (keseimbangan) [1].

Genre video game yang digunakan dalam penelitian ini adalah turunan dari *genre video game* FPS dan TPS. *First Person Shooter* (FPS) adalah *genre video game* yang menggunakan sudut pandang orang pertama dimana *player* benar-benar menjadi penembak itu sendiri. Contohnya seperti *game* counter strike, call of duty, dan *game* lainnya.



Gambar 1.1. *Game* Bergenre First Person Shooter (FPS)

Sementara, *Third Person Shooter* (TPS) adalah *genre* dari *game* yang menggunakan sudut pandang orang ketiga dimana *player* menggerakkan tokoh yang sudah disediakan oleh *game* itu sendiri. Contoh nya adalah *game* Grand Theft Auto (GTA). Dimana *player* adalah sosok *character* atau agen dalam *game* tersebut.



Gambar 1.2. *Game* Bergenre Third Person Shooter (TPS)

Keuntungan dari *game* genre FPS adalah *player* dapat merasakan bagaimana rasanya menjadi penembak yang dapat membunuh musuh dalam *game*, sekaligus dapat melatih kemampuan motorik atau refleks dalam menembak dan menghindar. Namun, kelemahan dari *genre game* ini adalah terlalu banyak jenisnya, karena *genre* dari *game* ini dianggap paling sukses dalam dunia perdagangan *game*, sehingga *player* biasanya menjadi jenuh karena terlalu sering memainkan *game* dengan *genre* FPS.

Sedangkan keuntungan dari *game* genre TPS adalah *player* bisa memperkirakan dimana *player* harus mengambil posisi untuk menyerang atau bertahan. Selain itu, *player* bisa berinteraksi dengan objek yang berada di sekitarnya seperti melompati barang, bertarung jarak dekat, atau menaiki kendaraan. Namun kelemahannya adalah *player* harus mempunyai akurasi yang tinggi dalam adu tembak dengan musuh, jelas kalah nyamannya dengan FPS jika harus beradu akurasi dalam adu tembak jarak dekat.

Game space shooter adalah sub *genre* dari FPS dan RTS. Dimana permainan tersebut mengharuskan *player* menghadapi musuh yang dikendalikan oleh *Non-Player Character* (NPC). Selama permainan berlangsung *player* dapat beradaptasi dengan perilaku menyerang dari NPC. Sehingga dengan berulangnya permainan dan perilaku musuh maka permainan menjadi membosankan karena perilaku NPC telah diketahui oleh *player*.



Gambar 1.3 (a) & (b) *Game Space Shooter*

Ketika *player* memulai suatu permainan, umumnya *player* lemah dan perlu adaptasi terhadap permainan, kemudian kemampuan *player* meningkat seiring adaptasi *player* terhadap rintangan yang ada dalam permainan, sehingga permainan menjadi membosankan karena *player* telah memahami perilaku menyerang dari NPC dalam permainan *space shooter*.

Salah satu hal yang dapat dilakukan pada permainan *space shooter* adalah dengan memberikan tantangan baru dan pengalaman yang berbeda pada permainan dengan memberikan suatu kemampuan adaptasi terhadap perilaku NPC seperti kemampuan adaptasi yang secara genetik telah dimiliki manusia sejak lahir. Kemampuan adaptasi pada manusia memiliki sejumlah keuntungan, terutama meningkatkan kemampuan seorang *player* untuk membiasakan diri menghadapi perilaku NPC.

Peneliti mencoba menerapkan kemampuan adaptasi yang dimiliki manusia kepada NPC. Dengan mengembangkan implementasi kecerdasan buatan dimana NPC dapat mengevolusi dirinya sendiri, sehingga mampu merespon perilaku *player*. Perkembangan kecerdasan buatan atau *artificial intelligent* (AI) terhadap NPC pada *game space shooter* masih banyak di kembangkan. *Artificial intelligent* dalam *game* bertujuan untuk membuat aksi dan reaksi secara otomatis pada perilaku NPC. Sehingga dapat memudahkan NPC dalam mengambil keputusan terbaik saat menghadapi *player*, sehingga *player* dapat menemukan kesan menyenangkan setelah memainkan permainan yang memiliki musuh dengan kemampuan adaptasi berupa evolusi dinamis NPC.

Sebuah *game* yang dirancang dengan baik dapat menggiring pemainnya untuk masuk ke zona *personal flow* yang dapat menumbuhkan perasaan senang dan

bahagia. Untuk membuat sebuah *game* menjadi menarik, Chen [6] menuturkan bahwa salah satu syarat awal yang dibutuhkan oleh pengembang teknologi adalah mengetahui apa yang diinginkan oleh user (dalam hal ini user adalah orang yang akan memainkan *game* atau sering disebut *player*)[6].

Csikszentmihalyi mengenalkan konsep *flow* [5], dalam usahanya membahas mengenai kebahagiaan. *Flow* direpresentasikan sebagai fokus yang dipusatkan pada sebuah kegiatan dengan tingkat perasaan bahagia dan perasaan utuh yang tinggi. Pada saat seseorang berada dalam zona *Flow*, dia dapat melupakan konsep mengenai waktu dan rasa cemas ketika bermain *game*. Inilah mengapa *game* yang dibangun berdasarkan konsep *Flow* membuat pemainnya tenggelam ke dalam permainan dan kerap melupakan waktu.

Dengan memahami konsep *flow* peneliti mencoba memberikan kemampuan adaptif pada NPC. Sehingga seiring waktu bersamaan dengan *player* meningkatkan kemampuannya dalam *game*, begitu pula dengan NPC yang dapat meningkatkan kemampuannya atau mengurangi kemampuannya tergantung dari perilaku *player*.

Sehingga dapat memunculkan keseimbangan dalam *game* antara kemampuan *player* dan musuh yang dihadapi. Pada umumnya, perilaku NPC dalam *game space shooter* memiliki *speed* dan jumlah *health* yang statis. Meskipun bos dalam *game space shooter* memiliki *health* yang cukup banyak dan *speed* yang cenderung rendah dengan kemampuan yang berbeda-beda pada jenis senjatanya.

Perilaku ini umumnya berupa musuh (NPC) yang memiliki aturan dan pola penyerangan yang telah diketahui sebelumnya oleh *player*. Sehingga *player* tidak perlu beradaptasi terhadap perilaku dari musuh (NPC). Hal ini membuat permainan terasa membosankan karena *player* telah memahami strategi yang diterapkan pada permainan *space shooter*. Sehingga peneliti mencoba memberikan kemampuan adaptasi terhadap NPC.

Kemampuan adaptasi yang dikembangkan pada NPC adalah evolusi dinamis pada *speed* dan *health* dari NPC pada setiap level, sehingga *player* akan menemukan kesenangan ketika bermain *game*, karena NPC dapat merubah kecepatan dan jumlah nyawanya secara dinamis sesuai dengan kemampuan *player* dengan menambahkan metode optimasi untuk menentukan *optimal solution* dari NPC pada setiap level permainan yang akan dihadapi *player*.

Dalam kamus Bahasa Indonesia, W.J.S. poerdwadarminta (1997 :753) dikemukakan bahwa : “Optimasi adalah hasil yang dicapai sesuai dengan keinginan, jadi optimasi merupakan pencapaian hasil sesuai harapan secara efektif dan efisien”. Optimasi juga banyak diartikan sebagai ukuran dimana semua kebutuhan dapat dipenuhi dari kegiatan-kegiatan yang dilaksanakan [2]. Menurut Winardi (1999 : 363) Optimasi adalah ukuran yang menyebabkan tercapainya tujuan sedangkan jika dipandang dari sudut usaha, optimasi adalah usaha memaksimalkan kegiatan sehingga mewujudkan keuntungan yang diinginkan atau dikehendaki. Dari uraian tersebut diketahui bahwa optimasi hanya dapat diwujudkan apabila dalam pewujudannya secara efektif dan efisien [3].

Untuk mengoptimasi dua atau lebih keadaan berbeda yang saling bertentangan inilah disebut dengan *Multi Objective Optimization*. *Multi Objective Optimization* (MOO) adalah suatu proses yang secara simultan mengoptimalkan dua atau lebih fungsi tujuan yang saling bertentangan dengan kendala yang ada (Gutierrez, 2012). Multi-objective Optimization dalam kasus nyata banyak digunakan untuk menangani permasalahan yang harus memenuhi lebih dari satu fungsi tujuan [4]. Untuk mengoptimasi MOO pada penelitian ini digunakan *Genetic Algorithm*.

Genetic Algorithm atau dalam bahasa Indonesia biasa disebut Algoritma Genetika adalah algoritma yang memanfaatkan proses seleksi alamiah yang dikenal dengan proses evolusi. Dalam proses evolusi, individu secara terus-menerus mengalami perubahan *gen* untuk menyesuaikan dengan lingkungan hidupnya. “Hanya individu-individu yang kuat yang mampu bertahan”. Proses seleksi alamiah ini melibatkan perubahan *gen* yang terjadi pada individu melalui proses perkembang-biakan. Dalam algoritma Genetika, proses perkembangbiakan menjadi proses dasar yang menjadi perhatian utama, dengan dasar berpikir: “Bagaimana mendapatkan keturunan yang lebih baik”.

Algoritma Genetika yang digunakan adalah *Non-dominated Sorting Genetic Algorithm II* (NSGA-II). NSGA-II merupakan algoritma genetika untuk optimasi multi-objektif berdasarkan non-dominasi. Variable keputusan diambil dari parameter *player* seperti *health player*, *score player*, *NPCMissed*, dan *damage player*. Variable keputusan tersebut digunakan untuk mencari *optimal solution*

untuk evolusi dinamis *health* dan *speed* dari musuh (NPC).

1.2 Rumusan Masalah

Agar permainan *space shooter* dapat memberikan tantangan baru dan pengalaman yang berbeda dengan mengimplementasikan kemampuan adaptasi kepada NPC (evolusi dinamis pada NPC) berdasarkan perilaku dari *player*. Sehingga permainan menjadi menyenangkan karena NPC memiliki kemampuan adaptasi terhadap perilaku *player* yang dapat membuat NPC dapat merubah *speed* dan *health* nya seiring permainan berlangsung.

1.3 Batasan Penelitian

Dalam penelitian ini, digunakan 4 macam pilihan parameter atau variable keputusan (*health player*, *score player*, *NPCmissed*, dan *damage player*) yang digunakan sebagai parameter untuk evolusi *speed* dan *health* dari NPC, dua tipe senjata yaitu gun dan rudal, tiga bonus item yang dapat diambil ketika *player* berhasil menghancurkan NPC, bonus item tersebut antara lain bonus item untuk menambah jumlah *weapon*, bonus item untuk mengaktifkan *shield* serta bonus item untuk menambah *health player*. Sedangkan bagi NPC hanya disediakan 4 jenis senjata yang dapat digunakan untuk menghancurkan pesawat *player*.

1.4 Tujuan Penelitian

Tujuan dalam penelitian tesis ini adalah sebagai berikut:

1. Mengembangkan *AI Game* yang dapat mengevolusi dinamis *speed* dan *health* NPC berdasarkan perilaku dan karakteristik dari *player* yang sedang dihadapi.
2. Perilaku dan karakteristik dari *player* digunakan sebagai parameter keputusan untuk evolusi dinamis *speed* dan *health* NPC pada level berikutnya dengan menggunakan NSGA-II.

1.5 Manfaat Penelitian

Manfaat dalam penelitian tesis ini adalah :

1. Mengembangkan penelitian lebih dalam pada bidang *AI Games* terutama *genre FPS* dan *TPS* pada *game space shooter*.
2. Menciptakan suatu *game* dengan tingkat kesulitan adaptif yang menitik beratkan pada evolusi dinamis *speed* dan *health* pada NPC berdasarkan kemampuan dari *player*. Sehingga *game* menjadi tidak membosankan karena NPC memiliki perilaku yang adaptif sesuai dengan kemampuan *player*

1.6 Kontribusi Penelitian

Pada penelitian ini, metode optimasi multi obyektif diterapkan untuk kecerdasan komputasional pada *Non Player Character* (NPC) untuk memberikan perlawanan kepada *player* berupa kemampuan adaptasi parameter yang disebut dengan evolusi dinamis *speed* dan *health* pada NPC. Diharapkan hasil penelitian dalam bentuk pengujian metode di dalam *game Space Shooter* dapat dijadikan penelitian berkelanjutan di dalam *game* dengan *genre FPS* yang lainnya. Selain itu, hasil penelitian ini dapat dijadikan pembandingan untuk penelitian yang melibatkan multi obyektif berikutnya.

1.7 Sistematika Penulisan

Penulisan buku tesis ini terbagi menjadi 5 bab yaitu Pendahuluan, Kajian Pustaka dan Dasar Teori, Metode Penelitian, Hasil dan Pembahasan, dan Kesimpulan. Penjelasan singkat mengenai masing-masing bab adalah sebagai berikut:

BAB I PENDAHULUAN

Bab ini menguraikan motivasi penelitian, permasalahan yang menjadi landasan untuk melakukan penelitian, perumusan dari topik permasalahan, serta tujuan dan manfaat dari penelitian yang dilakukan. Selain itu, dijelaskan pula metodologi penelitian yang digunakan serta sistematika laporan.

BAB II KAJIAN PUSTAKA

Bab ini terdiri dari Kajian Pustaka yang berisi referensi-referensi dari penelitian sebelumnya yang berkaitan dengan penelitian ini meliputi metode, algoritma, dan teknik pemrosesan data serta Dasar Teori yang menjelaskan tentang

pemahaman dasar dari penelitian ini meliputi topik yang menjadi permasalahan dan metode yang digunakan.

BAB III METODE PENELITIAN

Bab ini membahas test-bed yang digunakan untuk penelitian meliputi konseptual model dan desain berorientasi obyek dari test-bed. Implementasi teori-teori pada bab 2 akan dijelaskan pada bab ini disertai dengan perancangan optimasi model meliputi variabel keputusan, batasan, dan formula fungsi obyektif. Pencapaian penelitian akan dijelaskan lebih mendalam di bab ini.

BAB IV HASIL DAN PEMBAHASAN

Bab ini menjelaskan mengenai hasil yang diperoleh dari hasil penelitian mulai dari percobaan dengan modifikasi parameter, modifikasi skenario *game*, hingga modifikasi fungsi obyektif disertai dengan pembahasan hasil tersebut.

BAB V KESIMPULAN

Bab ini menjelaskan mengenai kesimpulan yang didapat dari hasil penelitian meliputi parameter, formula, dan hasil. Bab ini juga diberikan sub bab saran yang berisi hal-hal yang bisa dikembangkan untuk melanjutkan penelitian ini.

BAB II

KAJIAN PUSTAKA

Penelitian ini dilakukan karena suatu permasalahan pada *game* yang umumnya tidak memiliki kemampuan adaptasi terhadap kemampuan *player*. Agar tujuan penelitian ini lebih terarah, diberikan kajian pustaka dari penelitian-penelitian sebelumnya yang berkaitan dengan penelitian ini. Sub bab ini diberikan kajian pustaka mengenai *Theory of Fun* berdasarkan *Csikszentmihalyi's Flow Theory*, pengenalan mengenai *multimodal networks* saat mengevolusi perilaku untuk “*multitask domains*” (*Evolving Multimodal Network for Multitask Games*), evolusi dinamis *speed* dan *health* NPC dalam *game Space Shooter* dapat membuat *game* lebih adaptif terhadap kemampuan *player* sehingga *game* menjadi menyenangkan karena musuh (NPC) memiliki perilaku dimanis saat menghadapi *player* (Evolusi Dinamis Perilaku *Non-Player Character* Dalam Permainan *Space Shooter* Menggunakan NSGA II).

2.1 Kajian Penelitian Terkait

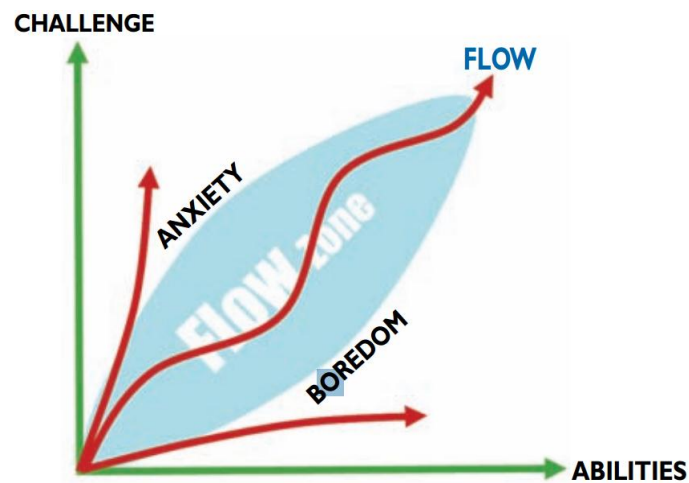
2.1.1 Csikszentmihalyi's Flow Theory

Sangat sulit untuk menjelaskan secara spesifik kesenangan yang dialami *player* ketika memainkan *game* video. Beragam teori dan definisi mengenai konsep *fun* telah diajukan. Untuk melakukan penelitian di dalam *game*, permasalahan ini menjadi hal yang penting untuk mendapatkan pemahaman bagaimana “*fun*” bisa terukur dan memberikan daya tarik untuk target *player*.

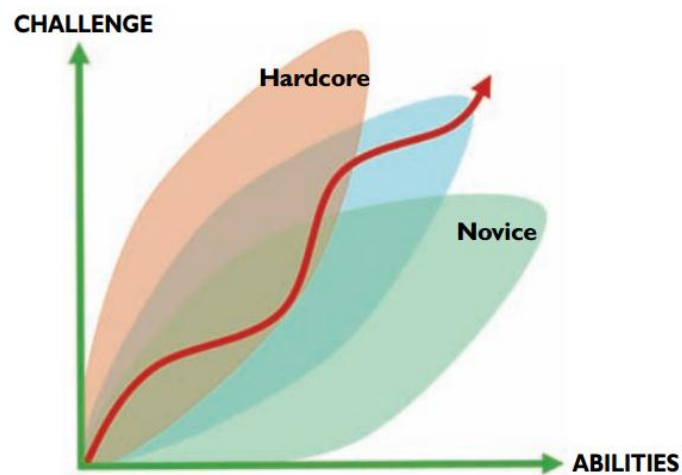
Csikszentmihalyi mengenalkan konsep *flow* [5] dimana ia menemukan bahwa orang-orang yang melakukan aktifitas yang mereka suka, memasuki “*state of immersion*” dan konsentrasi yang terfokus. Orang-orang yang menemukan kesenangan pada aktifitas tertentu mengilustrasikan bagaimana sekumpulan tantangan dan sekumpulan kemampuan pada pengalaman yang optimal.

Chen, J [6] menunjukkan bahwa deskripsi dari pengalaman “*flow*” itu identik dengan apa yang *player* alami di dalam *game*, kehilangan waktu, dan tekanan dari luar, beserta ketertarikan lainnya. Ia menunjukkan bahwa ini merupakan hal penting untuk menyamakan tantangan dengan kompetensi. Jika tantangan dalam *game* terlalu rendah, *player* akan menjadi bosan. Jika terlalu tinggi, *player* akan frustrasi (Gambar 2.1)

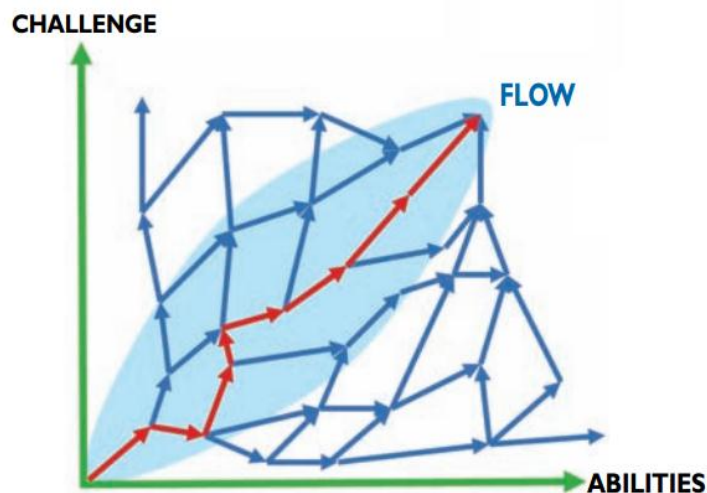
Sebagai tambahan, seorang *player* juga harus mempelajari *game*, sehingga ini menjadi hal yang penting untuk meningkatkan kemampuan secara progresif di dalam *game* untuk menjaga kondisi *flow* [7]. *Flow* menjadi teknik yang populer untuk mengukur “*fun*” dari suatu *game*, di mana pengembang *game* mengukur *gamenya* dengan cara *flow* untuk memberikan *player* pengalaman optimal. Bagaimanapun mengukur tantangan dan kompetensi untuk memberikan *flow* bukan hal yang mudah.



1. Zona *flow*



2. Zona *flow* dari *player* yang berbeda.



3. Desain adaptasi pengalaman

Gambar 2.1 Menunjukkan digram *flow* a) Grafik visual dari zona *flow*. b) *Player* yang berbeda memiliki zona *flow* yang berbeda. c) Desain adaptasi pengalaman dari *player* melalui pilihan *flow* yang sengaja dibangun oleh *player* dan menjadi pengalaman bagi *player*. [6]

Kiili, K. [8] memberikan model *game* eksperiensial untuk pengembang *game* agar dapat mengukur *flow* di dalam *gamenya* melalui menghitung karakteristik *flow* yang muncul pada *player*. Model ini diajukan untuk mendeskripsikan proses belajar di dalam *game* pada tingkat abstrak. Melalui fokus terhadap pengalaman *flow* dalam model *game*, kondisi *flow* dapat diukur dari karakteristik kunci *player* saat bermain seperti konsentrasi, distorsi waktu, pengalaman *autotelic*, kehilangan kesadaran, dan *sense of control*. Alternatif yang dapat diobservasi dari kondisi *flow* saat selesai bermain adalah perilaku menyelidik (*exploratory behaviour*) dan pembelajaran aktif (*active learning*).

Gilleade, K. dan Dix, A. [5] mengukur frustrasi fisik sebagai cara untuk mendesain permainan video adaptif. Melalui pengenalan tingkat frustrasi dalam *game*, perubahan yang diharapkan dapat diidentifikasi dan diperbaiki. Sama halnya di dalam konteks *game* adaptif dimana ini dapat mengimbangi kemampuan *player*. Penelitian menunjukkan bahwa mengukur umpan balik afektif dari *player* melalui pengukuran secara psikologis seperti tekanan darah dan detak jantung dianggap tidak baik ketika digunakan untuk lingkungan *game* tradisional.

Namun, mengukur frustrasi melalui perangkat masukan (*mouse, keyboard*) dan permainan itu sendiri (*game progress*) adalah solusi yang lebih baik. Sebagai tambahan,

data yang terekam dari *player* dapat beragam berdasarkan latar belakang individu. Penelitian ini menunjukkan pentingnya memperhatikan faktor luar dari lingkungan *game*, memberikan penelitian lebih jauh ke dalam *game* afektif untuk mengukur frustrasi secara akurat.

2.1.2 Evolving Multimodal Network for Multitask Games

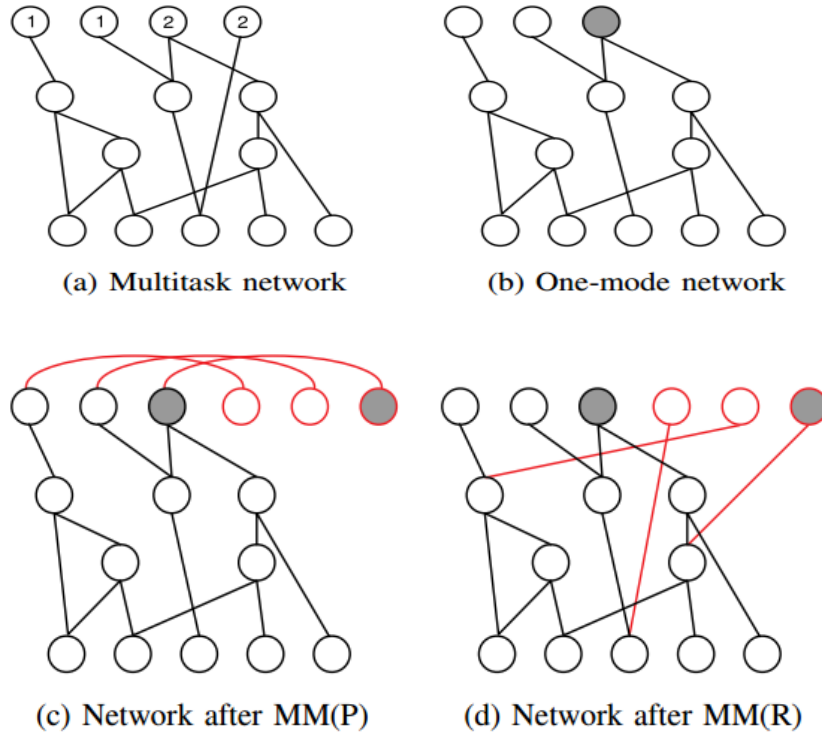
Risto Miikkulainen dan Jacob Schrum [8] menjelaskan mengenai manfaat dari *multimodal networks* dalam mengevolusi perilaku untuk “*multitask domain*”. Domain yang dimaksud di sini adalah ketika *agen* menghadapi beberapa tugas yang berbeda dalam evaluasi yang berbeda pula. Sebagai contoh, *agen* dievaluasi dalam satu tugas dan menerima nilai *fitnes* dan kemudian pada tugas yang berbeda akan mendapatkan nilai *fitnes* yang lain.

Nilai yang digabungkan dari semua tugas digunakan untuk mendefinisikan performa dari *agen*. Mengevolusi jaringan saraf *multimodal* membantu untuk menemukan *agen* yang mampu melakukan sejumlah tugas. Untuk mendapatkan sejumlah nilai dari sejumlah tugas, evolusi multiobyektif berdasarkan Pareto digunakan menyelesaikan masalah ini. Konsep ini disebut dengan *Neuroevolution* [9]. Terdapat dua metode yang digunakan untuk mengevolusi jaringan *multimodal* yaitu:

1. *Multitask Learning*: *Multitask learning* mengasumsikan bahwa *agen* selalu menyadari tugas yang dihadapi. Setiap jaringan dilengkapi dengan set lengkap dari neuron keluaran per tugasnya seperti yang ditunjukkan pada Gambar 2.2(a). Kemudian, jika dua keluaran dibutuhkan untuk mendefinisikan perilaku dari NPC dan NPC harus menyelesaikan dua tugas. Kemudian jaringan akan memiliki dua keluaran untuk setiap tugasnya dari total empat keluaran. Ketika melakukan tugas yang diberikan, NPC berperilaku berdasarkan keluaran pada tugas saat ini dan mengabaikan keluaran lainnya.
2. *Mode mutation*: *Mode mutation* tidak memberikan NPC pengetahuan mengenai tugas saat ini. Ini adalah tugas operator mutasi yang menambahkan mode output baru ke dalam jaringan. Hasilnya adalah jaringan dapat memiliki banyak mode keluaran berbeda dan sering kali melebihi jumlah tugas yang ada pada domain. Tidak ada pemetaan dari mode ke tugas untuk mendefinisikan perilaku NPC untuk setiap waktu yang dibutuhkan. Pengambilan keputusan berdasarkan neuron keluaran yang disebut neuron preferensi. Setiap mode memiliki neuron preferensi dengan tambahan beberapa neuron acuan seperti neuron yang menentukan perilaku

agen. Setiap waktu, mode keluaran yang mana neuron preferensinya paling tinggi akan dipilih. Jadi jika dua neuron dibutuhkan untuk mendefinisikan perilaku agen, *mode mutation* menambahkan tiga neuron ke lapis keluaran: dua neuron acuan dan satu neuron preferensi. Terdapat dua *mode mutation* dalam paper ini, yaitu:

- a. MM(P): MM(P) atau *Mode Mutation Previous* adalah *neuroevolution* dengan *mode mutation* yang mana mode baru terhubung ke mode sebelumnya. *Mode mutation* menambahkan set mode keluaran baru ke dalam jaringan. Setiap mode memiliki neuron preferensi, keluaran yang mana mengindikasikan preferensi dari jaringan (cenderung ke mode lainnya) untuk menggunakan mode tersebut dalam waktu evaluasi. Jaringan dapat belajar untuk memilih antara mode yang berbeda pada situasi yang berbeda, yang mana dapat menolong untuk mengembangkan mode berbeda pada tugas yang berbeda. Dengan tipe mode mutation seperti ini, semua mode masukan baru datang dari mode jaringan yang ada sebelumnya, untuk menjamin kesamaan dengan mode yang telah ada. Jaringan MM(P) ditunjukkan pada Gambar 2.2(c).
- b. MM(R): MM(R) atau *Mode Mutation Random* adalah mode mutation yang mana masukan baru datang dari lapis tersembunyi yang berbeda ke dalam jaringan. Pendekatan ini cenderung untuk membuat mode baru yang sangat berbeda dari mode yang ada, yang mana membantu untuk mempercepat perilaku baru. MM(R) juga mendukung dan menggunakan mode penghapusan mutasi, yang mana menghapus mode yang paling jarang digunakan dari evaluasi sebelumnya, namun operasi ini selalu aman pada MM(R). Jaringan MM(R) ditunjukkan pada Gambar 2.2(d).



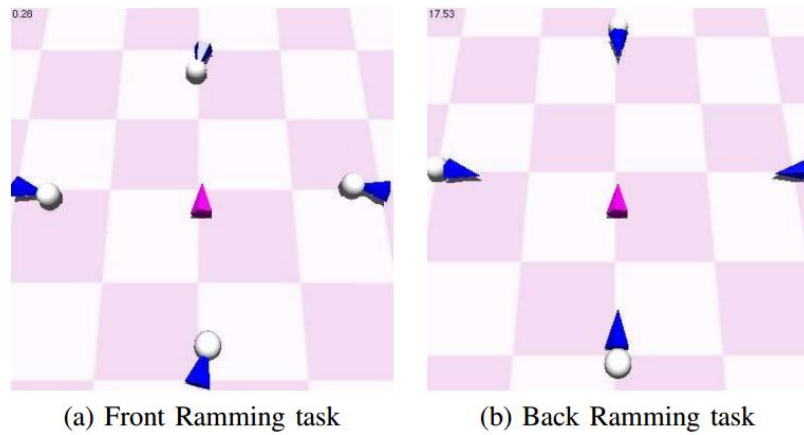
Gambar 2.2 Jaringan untuk memainkan *game* multitask [2]. (a) Jaringan multitask dengan dua node, setiap node terdiri dari dua output. (b) Jaringan dengan satu-mode keluaran mengandung neuron preferensi abu-abu. (c) Modifikasi jaringan satu-mode oleh MM(P). (d) Modifikasi jaringan satu-mode oleh MM(R).[6]

Metode tersebut dievaluasi ke dalam dua domain. Setiap domain memiliki dua tugas yang berbeda. Berikut adalah domain yang digunakan untuk evaluasi metode tersebut:

1. *Front/Back Ramming Domain*

Untuk setiap tugas, *Evolved agent* memiliki *ram* (bola putih) yang digunakan sebagai senjata untuk menyerang *scripted agent*. *Evolved agent* juga rentan terhadap serangan *scripted agent*, yang mana bergerak menghindari *ram* untuk menyerang *evolved agent*. Perbedaan antara dua tugas adalah posisi dari *ram*. Pada tugas *Front Ramming*, *ram* berada di depan *evolved agent* dan tugas *Back Ramming*, *ram* berada di belakang *evolved agent*. *Evolved agent* harus mempelajari mengenai orientasi diri terhadap tugas sehingga mereka dapat

menggunakan *ram* dengan benar. *Game Front/Back Ramming* ditunjukkan pada Gambar 2.3.

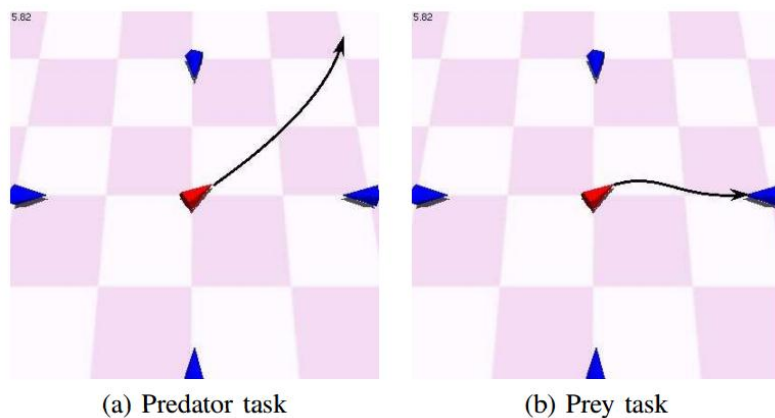


Gambar 2.3 *Game Front/Back Ramming*. (a) Posisi awal tugas front ramming, (b) Posisi awal tugas back ramming.

Dalam *Front/Back Ramming* NPC mulai menunjuk musuh di tengah. Domba-domba jantan digambarkan oleh bola putih menempel pada NPC. Dalam tugas front ramming, NPC dapat mulai menyerang musuh segera, tetapi dalam tugas back ramming mereka harus berbalik terlebih dahulu.

2. *Predator/Prey Domain*

Pada tugas *Predator*, suatu tim yang terdiri dari empat *evolved agent* mencoba untuk menyerang *scripted prey agent* sambil mencoba untuk mencegahnya kabur. Pada tugas *Prey*, tim yang terdiri dari empat *evolved agent* harus berusaha kabur dari *scripted agent* yang sama, yang mana sekarang berperilaku sebagai predator. *Game Predator/Prey* ditunjukkan pada Gambar 2.4



Gambar 2.4 *Game Predator/Prey*. Kedua tugas *Predator* dan *Prey* terlihat sama. (a) jalur pergerakan musuh dalam tugas *Predator*: Akan mencoba untuk melarikan diri melalui celah terdekat antara dua NPC. (b) Jalur musuh dalam tugas *Prey*: mengejar mangsa NPC terdekat di depannya.

Kedua situasi terlihat sama dengan NPC, tetapi karena dinamika lingkungan dan perilaku musuh yang berbeda, perilaku yang berbeda diperlukan agar berhasil.

2.1.3 Evolusi Penyerangan Creep

Ariyadi [10] dalam penelitiannya mengungkapkan ide untuk membuat sebuah *game* yang adaptif terhadap strategi *player* dalam *game* Tower Defense. Dalam *game* Tower Defense terdapat serangan NPC yang dikendalikan oleh *Non-Player Character* (NPC) statis dimana *player* akan terus berusaha beradaptasi terhadap strategi yang digunakan. Pada akhirnya, *game* akan menjadi membosankan karena kemampuan yang jauh di bawah *player* atau membuat frustrasi jika NPC terlalu sulit.

Ariyadi menggunakan *Dynamic Difficulty Adjustment* (DDA) untuk mengatur evolusi penyerangan NPC dengan cara memeriksa perbandingan jumlah NPC yang mati dengan jumlah NPC yang dibangkitkan. Kemudian *Non-Dominated Sorting Genetic Algorithm II* (NSGA-II) digunakan untuk mencari solusi terbaik untuk menjadi komposisi NPC berikutnya yang telah diatur tingkat kekuatannya oleh DDA agar mampu bertahan dari serangan tower yang dipasang oleh *player*.

Dengan menggunakan DDA dengan metode NSGA-II dihasilkan nilai kontrol DDA dengan peningkatan antara 1 hingga 1.5 kali, di mana 1.5 berarti *player* berhasil menghabiskan keseluruhan NPC. NSGA-II menghasilkan solusi optimal dalam bentuk grafik dengan 3 obyektif. Berdasarkan hasil simulasi, kestabilan berada pada generasi ke-20 dengan parameter populasi=100, probabilitas persilangan (p_c) = 1, probabilitas mutasi (p_m)= $1/n$, indeks distribusi persilangan (η_c)=20, dan indeks distribusi mutasi (η_m)=20.

2.1.4 Perilaku NPC Saat Pertahanan Tempur

Rahmat Fauzi [11] dalam penelitiannya mengembangkan kecerdasan buatan atau *Artificial Intelligent* (AI) NPC pada *game Real Time Strategi* (RTS). AI NPC terus dikembangkan agar NPC menjadi cerdas dan bisa menirukan perilaku manusia. Salah satu penelitian tentang perilaku AI NPC adalah perilaku kecerdasan NPC dalam strategi bertahan. Dalam bagian pertahanan terdapat permasalahan yakni NPC pertahanan memilih musuh yang terdekat atau musuh yang pertama kali memasuki wilayah pertahanan. Padahal, musuh tersebut belum pasti lebih berbahaya atau lebih kuat. Oleh karena itu, diperlukan kecerdasan agar NPC pertahanan lebih kuat serta NPC lebih cerdas dalam memilih musuh dan menyesuaikan perilaku untuk menghadapi jenis musuh.

Rahmad Fauzi memodelkan strategi bertahan dengan melihat perilaku NPC dengan menggunakan *Hierarchical Finite State Machine* (HFSM). Metode *Hierarchical Finite State Machine* (HFSM) adalah salah satu metode untuk mensimulasikan perilaku NPC. Metode HFSM merupakan metode yang terdiri dari beberapa *state* dan juga beberapa kumpulan *state* yang biasa disebut Hierarki *State*. HFSM ini berfungsi untuk perpindahan keputusan NPC dari satu *state* ke *state* berikutnya. Dengan harapan menemukan sebuah perilaku pertahanan yang lebih kuat dan lebih selektif terhadap musuh.

2.2 Dasar Teori

Dasar Teori menjelaskan mengenai teori-teori yang berkaitan dengan penelitian. Ini telah diurutkan untuk memberikan gambaran penelitian dan menangkap *state-of-play* saat ini. Berikut adalah Dasar Teori mengenai topik penelitian yang dilakukan.

2.2.1 Space Shooter

Space Shooter adalah *subgenre* dari *game* video bergenre *First Person Shooter* (FPS). Tujuan utama dari *game* ini adalah menghancurkan pesawat musuh atau menghindari serangan musuh hingga mencapai garis akhir atau mengalahkan bos disetiap level.

Dalam *game Space Shooter* terdapat satu atau beberapa *player* dan terdapat

banyak sekali NPC yang harus dihadapi oleh *player* sebelum mencapai garis akhir. NPC dalam *game Space Shooter* memiliki kemampuan untuk mengikuti pergerakan dari *player* dan dapat menembakkan senjata kearah *player*.

Setiap kali *player* dapat menghancurkan musuh (NPC) maka *player* akan mendapatkan hadiah berupa point, dan jika *player* berhasil menghancurkan NPC yang terdapat item didalamnya maka *player* dapat mengambil hadiah item berupa *armor*, *update weapon*, dan *health*. Dan jika NPC berhasil menembak pesawat *player* hingga *health player* habis, maka *player* dinyatakan kalah dan harus mengulang *game*.



Gambar 2.5 Contoh *game Space Shooter*

2.2.2 *Game Play* Pada Permainan

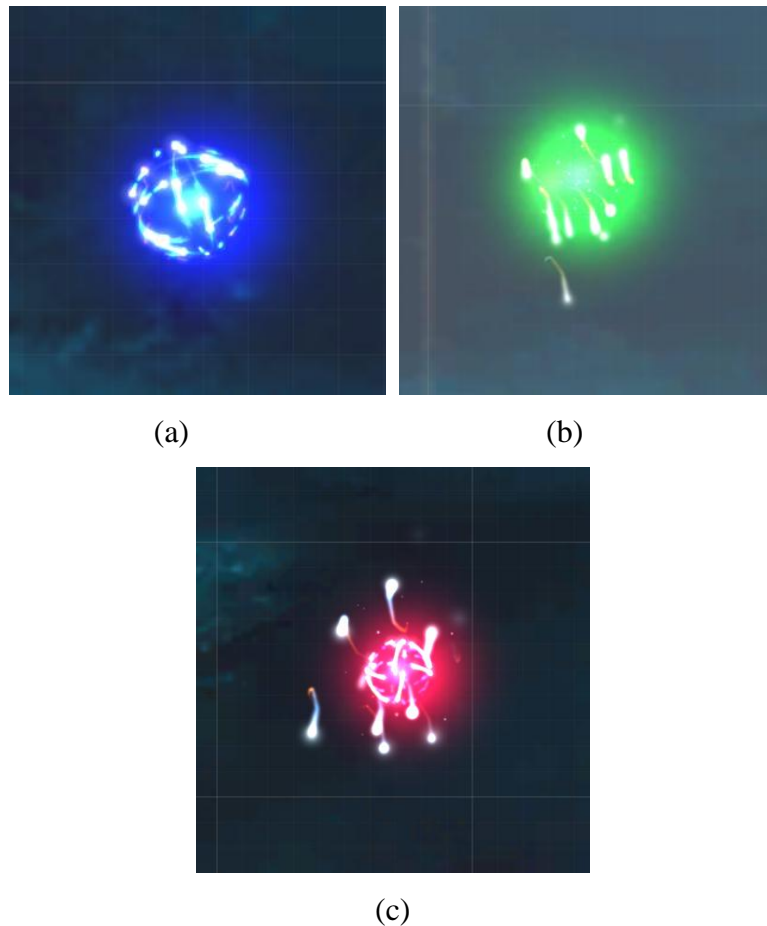
Gameplay adalah bagaimana cara seorang *gamer* untuk berinteraksi dengan *game* yang sedang dimainkannya. Gameplay dalam sebuah *game* sangatlah berpengaruh dalam pembuatan *game* maupun dalam memainkan *game*. Jika sebuah *game* memiliki *gameplay* yang mudah untuk dimainkan maka semakin banyak *player* yang menyukainya, walaupun kadang kala ada beberapa *game* yang memiliki *gameplay* yang tidak mudah untuk dimainkan tapi masih dapat menarik minat *player* untuk memainkan *game* tersebut.

Dalam penelitian ini terdapat *gameplay* yang mudah dimengerti karena *player* hanya menggerakkan pesawat untuk naik, turun, kiri, dan kanan. Serta disediakan dua jenis senjata yang dimiliki oleh pesawat *player*, antara lain *gun* dan *rudal*.

Selama permainan berlangsung, *player* harus dapat menghancurkan pesawat musuh hingga bertemu bos, jika *player* dapat menghancurkan pesawat bos maka *player* berhak naik level. Namun jika pesawat *player* hancur karena ditembak NPC atau hancur ketika menghadapi bos, maka permainan harus dimulai dari awal lagi dengan scor 0, *health* 100, NPC miss 0 dan damage 0.

Dalam pesawat musuh terdapat tiga item yang dapat mendukung kemungkinan *player* untuk menang. Tiga item tersebut antara lain item hijau (*health*), item merah (*shield*), dan item biru (rudal).

Masing-masing item memiliki fungsinya masing-masing, seperti item hijau adalah item untuk *health*, dimana setiap kali *player* berhasil mendapatkannya maka *health* pesawat *player* dapat bertambah 20. Item merah adalah item untuk mengaktifkan *shield*, dimana setiap kali *player* berhasil mendapatkannya maka pesawat *player* dapat terlindungi dari semua serangan musuh. Item biru adalah item untuk *rudal*, setiap kali *player* berhasil mendapatkannya maka senjata *rudal* dari pesawat *player* dapat bertambah 1 dalam stock senjata rudal *player*.




Gambar 2.6 Item Dengan Efek *Lightning* (a) Item Untuk Menambah satu Rudal Dalam Stock senjata (b) Item Untuk Menambahkan *health player* (c) Item Untuk Mengaktifkan *Shield Player*



Untuk mendapatkan item tersebut *player* harus dapat menghancurkan pesawat musuh agar item tersebut dapat terjatuh dari pesawat musuh, selain itu *player* harus mengambil item yang terjatuh sebelum item tersebut menghilang. Tidak semua pesawat musuh yang berhasil dihancurkan oleh *player* terdapat item didalamnya. Oleh karena itu permainan dapat menjadi semakin menarik karena *player* tidak dapat menerka pesawat mana yang terdapat item didalamnya.

Dalam gameplay yang dibuat, desain senjata dan item yang dimiliki oleh pesawat *player* dan pesawat NPC sengaja dibuat berbeda agar terdapat keseimbangan kemampuan antara pesawat *player* dan pesawat NPC. Sehingga tidak terjadi tumpang tindih kemampuan yang menyebabkan permainan menjadi membosankan.

Selama permainan berlangsung pesawat *player* dibekali dua jenis senjata, yaitu gun dan rudal, serta tiga macam item yang dapat diperoleh oleh *player* jika *player* dapat menghancurkan musuh dan mengambil item yang terjatuh. Pada tabel 2.1 dijelaskan secara rinci spesifikasi pada pesawat *player*. Mulai dari nama-nama item dan senjata hingga fungsinya selama permainan.

Tabel 2.1 Jenis – Jenis Senjata dan Item *Player*

Item	Fungsi
	Item Untuk Menambah satu Rudal Dalam Stock Senjata
	Item Untuk Menambahkan <i>health player</i>
	Item Untuk Mengaktifkan <i>Shield Player</i>

	Senjata gun yang digunakan <i>player</i> untuk menghadapi musuh
	Senjata rudal yang digunakan <i>player</i> untuk menghadapi musuh




Spesifikasi yang dimiliki oleh pesawat *player* dapat dilihat pada tabel 2.2. Dimana terdapat *speed* dari *player*, *health* dari *player*, macam-macam item dan pengaruh item tersebut terhadap pesawat *player* selama dalam permainan serta jenis senjata dan *damage* yang dimiliki oleh senjata dari pesawat *player*.

Tabel 2.2 Spesifikasi Pesawat *Player*

<i>Speed Player</i>	<i>Health Player</i>	<i>Item</i>	<i>Fungsi Item</i>	Jenis Senjata <i>Player</i>	<i>Damage Senjata Player</i>
10	100	<i>Health</i>	+10	Gun	10
		<i>Weapon</i>	+1		
		<i>Armor</i>	10 s	Rudal	30

Namun tidak hanya pesawat *player* yang didesain sedemikian rupa sehingga lebih menarik. Karena selama permainan berlangsung, pesawat dari *non-player character* (NPC) juga didesain dengan lengkap agar dapat menjadi musuh yang seimbang bagi *player*. Oleh karena itu dibuat beberapa variasi senjata yang cukup banyak untuk pesawat NPC. Karena desain gameplay pada pesawat NPC tidak disertakan item seperti yang dimiliki oleh *player*. Pesawat NPC hanya memiliki senjata yang dapat digunakan untuk menghadapi pesawat *player* selama permainan berlangsung seperti yang terlihat pada tabel 2.3.

Tabel 2.3. Macam-macam Senjata NPC

Nama Senjata	Fungsi
	Senjata <i>gun</i> yang digunakan NPC untuk menghadapi <i>player</i>
	Senjata <i>missile</i> yang digunakan NPC untuk menghadapi <i>player</i>
	Senjata rudal yang digunakan NPC untuk menghadapi <i>player</i>
	Senjata <i>suicide</i> yang digunakan NPC untuk menghadapi <i>player</i>

Spesifikasi yang dimiliki oleh pesawat NPC dapat dilihat pada tabel 2.4. Dimana terdapat *speed* dari NPC, *health* dari NPC, macam-macam jenis senjata dan *damage* yang dimiliki oleh senjata dari pesawat NPC. Sehingga tidak terjadi ketidakseimbangan antara kemampuan pesawat *player* dan pesawat NPC.

Tabel 2.4. Spesifikasi Pesawat NPC

<i>Speed</i> NPC	<i>Health</i> NPC	Jenis Senjata NPC	<i>Damage</i> Senjata NPC
10	100	Gun	10
		Missile	20
		Rudal	30
		Suicide	40

Terdapat 10 level uji coba pada penelitian ini, dimana pada setiap levelnya kromosom NPC dapat selalu berubah-ubah sesuai dengan perilaku *player*. Akan ada peningkatan maupun penurunan kemampuan dari NPC sesuai dengan parameter yang dimiliki oleh pesawat *player* yang didapatkan pada level sebelumnya untuk dapat mengevolusi parameter NPC. Parameter tersebut antara lain *health* dan *speed* dari pesawat NPC.

Namun pada level pertama pada *game space shooter*, ke empat varian senjata akan keluar dengan *speed* dan *health* yang masih konstan sesuai pada tabel 2.5. Dikarenakan pada level pertama, hanya sebagai data test terhadap perilaku *player*, sehingga pada level kedua hingga level ke sepuluh, nilai konstan dari *speed* dan *health* dari NPC dapat berubah-ubah sesuai perilaku *player*.

Penelitian ini dibuat untuk menentukan evolusi dinamis NPC pada level berikutnya dengan melihat parameter dari *player* pada level sebelumnya. Parameter tersebut antara lain *score player*, *health player*, *NPCmissed*, dan *damage player* yang digunakan untuk menghancurkan NPC. Parameter tersebut digunakan untuk menentukan evolusi dinamis NPC pada level berikutnya.

Selama permainan berlangsung, parameter dari NPC seperti *speed* (f_1) dan *health* (f_2) dapat berubah sesuai dari parameter *player*. Karena dua *gen* ini digunakan untuk evolusi dinamis NPC untuk menghadapi *player*.

Dimana pada state awal *game*, *spawn* awal NPC dapat dianalogikan sebagai *default*, sebagai ujicoba tingkat kemampuan *player* pada level pertama dalam permainan. Setelah level pertama selesai, maka parameter yang ada pada *player*

(*score player*, *health player*, *NPCMissed*, dan *damage player*) dihitung untuk menentukan evolusi dinamis NPC, dimana NPC yang dibangkitkan pada level selanjutnya menggunakan senjata *gun*, *missile*, *rudal*, atau *suicide* beserta evolusi parameter pada NPC (f_1 dan f_2) .

2.2.3 Perilaku

Perilaku adalah respon individu terhadap suatu stimulus atau suatu tindakan yang dapat diamati dan mempunyai frekuensi spesifik, durasi dan tujuan baik disadari maupun tidak. Perilaku merupakan kumpulan berbagai faktor yang saling berinteraksi. Sering tidak disadari bahwa interaksi tersebut amat kompleks sehingga kadang-kadang tidak sempat memikirkan penyebab anak menerapkan perilaku tertentu. Karena itu amat penting untuk dapat menelaah alasan dibalik perilaku individu anak. Harlen (Sjarkawi, 2006: 35) mengemukakan bahwa :” perilaku merupakan kesiapan atau kecenderungan seseorang untuk bertindak dalam menghadapi suatu objek atau situasi tertentu”[12].

2.2.3.1 Adaptasi

Adaptasi adalah bagaimana organisme mengatasi tekanan lingkungan sekitarnya untuk bertahan hidup. Organisme yang mampu beradaptasi akan bertahan hidup, sedangkan yang tidak mampu beradaptasi akan menghadapi kepunahan atau kelangkaan jenis. Adaptasi adalah, pertama-tama, proses, dan bukan bagian fisik dari tubuh. Perbedaan dapat dilihat dalam parasit internal, dimana struktur tubuh sangat sederhana, tapi tetap organisme. Yang sangat beradaptasi dengan lingkungan yang tidak biasa.

Dari sini dapat dilihat adaptasi yang tidak hanya masalah sifat terlihat: dalam parasit seperti adaptasi kritis terjadi dalam siklus-hidup, yang sering cukup rumit. Namun, sebagai istilah praktis, adaptasi sering digunakan untuk. produk: fitur-fitur dari spesies yang hasil dari proses tersebut. Banyak aspek dari hewan atau tanaman dapat benar adaptasi disebut, meskipun selalu ada beberapa fitur yang fungsinya diragukan.

Dengan menggunakan istilah adaptasi untuk proses evolusi, dan sifat adaptif untuk bagian tubuh atau fungsi (produk), dua indera kata mungkin dibedakan.

Adaptasi adalah salah satu dari dua proses utama yang menjelaskan beragam spesies yang kita lihat dalam biologi, seperti berbagai jenis kutilang Darwin. Yang lainnya adalah spesiasi (spesies-membelah atau *cladogenesis*), yang disebabkan oleh isolasi geografis atau mekanisme lain. Sebuah contoh favorit digunakan sekarang untuk mempelajari saling adaptasi dan spesiasi adalah evolusi ikan cichlid di danau Afrika, mana pertanyaan isolasi reproduksi jauh lebih kompleks.

2.2.3.2 Adaptasi Perilaku Evolusi dinamis

Untuk memberikan perlawanan yang sepadan. NPC dalam penelitian ini diberikan kemampuan untuk beradaptasi dengan perilaku *player*. Sehingga NPC dapat mengevolusi *speed* dan *health* secara dinamis tergantung perilaku *player*.

Adaptasi yang dimaksud dalam penelitian ini adalah teori adaptasi fisiologi dan adaptasi tingkah laku. Dimana adaptasi fisiologi adalah penyesuaian diri makhluk hidup dengan cara melakukan proses fisiologis dalam tubuhnya agar dapat menjaga kelangsungan hidupnya. Yang berarti NPC harus dapat merubah bentuknya agar dapat bertahan hidup. Seperti merubah *health* nya secara dinamis untuk dapat menyamai kemampuan *player*.

Selanjutnya adalah adaptasi tingkah laku yang merupakan tingkah laku makhluk hidup untuk menyesuaikan diri dengan lingkungannya agar tetap bertahan hidup. Beberapa jenis hewan ada yang menyesuaikan diri dengan lingkungan dengan cara mengubah tingkah laku. Cara ini selain untuk mendapatkan makanan juga untuk melindungi diri dari musuh atau pemangsa. Yang dapat diimplementasikan dalam NPC pada permainan *space shooter* dengan merubah *speed* NPC sehingga *player* akan menemukan kesenangan karena NPC memiliki perilaku yang berubah-uban, sehingga *game* menjadi lebih menarik dan sesuai dengan mood dari *player*.

Evolusi perilaku dinamis *speed* dan *health* pada NPC yang dimaksud adalah jika *player* sedang serius bermain *game* dan memenangkan level dalam *game* dengan tetap mempertahankan *hp* yang tinggi, *score* yang tinggi, *NPCmissed* sedikit dan *damage* yang tinggi maka pada level selanjutnya NPC dapat merubah *speed* dan *health* nya dengan maksimal. Bila level berikutnya *player* kesulitan mengalahkan NPC dan cenderung tidak mampu menghadapi *player*, maka NPC dapat kembali merubah *speed* dan *health* nya agar dapat menyamai kemampuan

player. Sehingga *player* tidak dapat merasa kesulitan maupun terlalu mudah untuk menghadapi NPC

2.2.4 Multi Objective Optimization

Multi-Objective Optimization merupakan metode penyelesaian persoalan optimasi yang terdiri lebih dari satu fungsi tujuan secara simultan (bersamaan). Dalam pemecahannya, seringkali terjadi konflik diantara tujuan dan solusinya bukanlah solusi tunggal tetapi berupa himpunan solusi. Secara umum, bentuk matematis dari multiobjective optimization adalah :

$$\min\{f_1(x), f_2(x), \dots, f_k(x)\} \quad (2.1)$$

dimana k adalah jumlah dari fungsi tujuan. Simbol $\min \{\dots\}$ menandakan bahwa semua tujuan diminimalkan secara bersamaan. Algoritma *Genetika* yang telah merupakan metode optimasi dengan obyektif tunggal. Metode tersebut digunakan untuk masalah optimasi skalar karena fungsi obyektif selalu mencapai nilai optimal global tunggal atau skalar. Untuk multi obyektif, fungsi multi-obyektif membentuk vektor sehingga biasa disebut dengan optimasi vektor [13].

Setiap permasalahan optimasi multi obyektif dapat dituliskan sebagai berikut:

$$\underset{x \in \mathbb{R}^d}{\text{Minimize}} f(x) = [f_1(x), f_2(x), \dots, f_M(x)] \quad (2.2)$$

subject to

$$g_j(x) \leq 0, j = 1, 2, \dots, J, \quad (2.3)$$

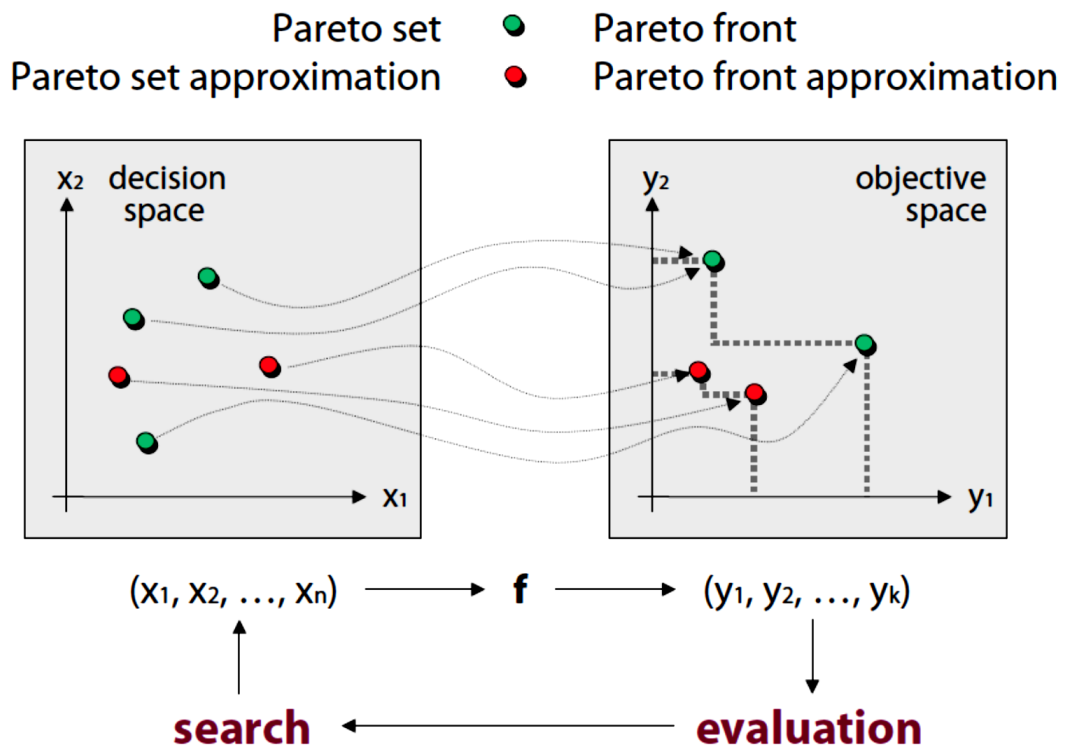
$$h_k(x) = 0, k = 1, 2, \dots, K, \quad (2.4)$$

dimana $x = (x_1, x_2, \dots, x_d)^T$ adalah vektor dari variabel keputusan. Pada beberapa formulasi yang digunakan pada literatur optimasi, pertidaksamaan g_j dimana $j = 1, \dots, J$ dapat pula dimasukkan ke dalam persamaan, karena persamaan $\phi(x) = 0$ dapat dikonversi ke dalam pertidaksamaan $\phi(x) \leq 0$ dan $\phi(x) \geq 0$. Namun untuk memperjelas, persamaan dan pertidaksamaan dipisahkan.

Ruang $\mathcal{F} = \mathbb{R}^d$ diisi oleh vektor dari variabel keputusan x yang biasa disebut ruang pencarian (*search space*). Ruang $S = \mathbb{R}^M$ dibentuk oleh semua nilai yang memungkinkan dari fungsi obyektif atau biasa disebut ruang obyektif (*objective space* atau *solution space*). Ruang pencarian dan ruang obyektif

ditunjukkan pada Gambar 2.6. Dibanding dengan fungsi obyektif tunggal yang mana ruang solusi adalah \mathfrak{R} , ruang solusi untuk optimasi multi obyektif memiliki ukuran yang lebih besar. Sebagai tambahan, saat dihadapkan dengan sebuah fungsi obyektif $f(x) = [f_i]$, untuk lebih mudah dapat menuliskan f_i sebagai $f(x)$ tanpa menyebabkan kebingungan.

Permasalahan optimasi multi obyektif tidak seperti permasalahan optimasi obyektif tunggal, tidak membutuhkan solusi optimal yang meminimalisasi semua fungsi multi obyektif secara simultan. Seringkali terjadi konflik antara obyektif satu dengan yang lain dan parameter optimal dari obyektif biasanya tidak mengarahkan kepada optimalitas obyektif lainnya (terkadang membuat lebih buruk). Sebagai contoh, sebuah keluarga yang menginginkan pelayanan kelas satu saat liburannya, namun menginginkan harga yang murah. Pelayanan tingkat tinggi (satu obyektif) akan sangat memakan biaya, namun ini menyebabkan konflik dengan obyektif lainnya (untuk meminimalisir biaya).



Gambar 2.7. Ruang Keputusan dan Obyektif (*Decision and Objective Space*)

Namun, di antara obyektif yang saling terjadi konflik, pemilihan *trade-off* harus dilakukan atau membentuk suatu keseimbangan pada obyektif. Jika tidak ada yang mungkin, daftar preferensi harus dipilih sebagai obyektif mana yang harus

didapatkan lebih dulu. Lebih penting lagi, obyektif yang berbeda harus dibandingkan dan dibuat suatu keputusan. Hal ini biasanya membutuhkan reformulasi dan salah satu pendekatan yang paling populer untuk reformulasi adalah menemukan fungsi bernilai skalar yang merepresentasikan kombinasi bobot atau urutan preferensi dari semua obyektif. Fungsi skala seperti ini sering kali disebut sebagai fungsi preferensi atau fungsi kegunaan (*utility function*). Cara yang mudah untuk membangun fungsi skalar adalah menggunakan jumlah tertimbang (*weighted sum*).

$$\Phi(f_1(x), \dots, f_M(x)) = \sum_{i=1}^M w_i f_i(x), \quad (2.5)$$

dimana w_i adalah koefisien penimbang.

Secara naif, beberapa pasti berpikir mengenai apa yang terjadi jika mencoba untuk mengoptimasi setiap obyektif secara individu sehingga setiap obyektif akan mendapatkan hasil terbaik (minimum untuk permasalahan minimalisasi). Dalam kasus ini, didapatkan:

$$F = (f_1^*, f_2^*, \dots, f_M^*), \quad (2.6)$$

yang biasa disebut dengan vektor obyektif ideal. Bagaimanapun tidak ada solusi yang mengarahkan kepada vektor ideal ini. Hal ini dapat dikatakan solusi yang tidak pernah ada (*nonexistent solution*). Satu-satunya pengecualian adalah ketika semua obyektif mengarah kepada solusi yang sama dan dalam kasus ini multi obyektif tidak mengalami konflik dan mengarahkan kepada kasus dimana Pareto front secara tipikal menuju ke *single point* [14].

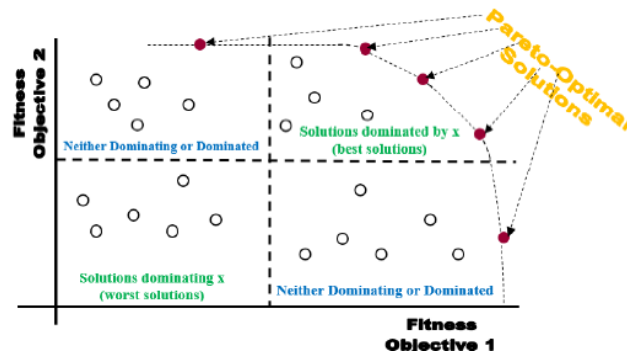
Tren terbaru saat ini dalam menyelesaikan permasalahan multi obyektif adalah dengan pendekatan evolusiuner seperti algoritma *genetika*.

2.2.5 Pareto Optimal

Sebuah solusi bisa saja yang terbaik, terburuk dan juga tidak berhubungan terhadap solusi lainnya (baik dominating atau dominated) dengan tetap memperhatikan sebuah nilai obyektif. Solusi terbaik dapat diartikan sebuah solusi tidak buruk pada obyektif manapun dan paling tidak satu obyektif lebih baik dari pada yang lainnya. Sebuah solusi optimal adalah solusi yang tidak didominasi oleh solusi manapun dalam ruang pencariannya. Seperti sebuah solusi optimal dikatakan Pareto – Optimal dan keseluruhan kumpulan dari seperti solusi optimal *trade-offs*

dikatakan Pareto-optimal set. Sebagai bukti, dalam situasi yang nyata keputusan (*trade-off*) adalah proses yang diperlukan untuk mendapatkan solusi yang optimal.

Walaupun terdapat beberapa cara untuk pendekatan persoalan multiobjektif optimization, sebagian besar pekerjaan terkonsentrasi pada perkiraan dari kumpulan pareto (Ajith Abrahan and Golberg 2005).



Gambar 2.8 Konsep Pareto Optimal

Multiobjective algoritma yang digunakan adalah NSGA II (Non Sort Genetic Algoritma II). Dasar dari NSGA II adalah proses eksekusi ranking sebelum operasi seleksi, Proses ini diidentifikasi sebagai solusi non dominan di dalam populasi, ditiap – tiap generasi ke bentuk *front nondominan*. Kemudian setelah itu adalah seleksi, *crossover*, dan mutasi. Didalam prosedur ranking yang pertama kali diidentifikasi adalah individu yang *non dominan* didalam populasi kemudian individu tersebut diasumsikan sebagai *front non dominan* yang pertama dengan nilai fitness yang paling besar untuk sementara [15].

Fitness yang nilainya sama diassign ke dalam *non dominan* tersebut untuk memperbaiki perbedaan. Didalam populasi digunakan metode *sharing* kemudian setelah itu individu dari *front* yang pertama sementara diabaikan kemudian populasi diproses lagi dengan cara yang sama untuk mengidentifikasi individual untuk *front nondominan* yang kedua. Sebuah nilai fitness sementara yang dishare dari *front* sebelumnya yang diassign ke semua individu yang mempunyai *front* yang baru. Proses ini dilanjutkan sampai semua populasi terklarifikasi ke dalam *non dominan front*.

Ketika *non dominan front* teridentifikasi kemudian populasi yang dihasilkan adalah menurut nilai fitness sementara tersebut. NSGA pertama kali menggunakan *procedure stochastic remainder propotional selection* (SRS). Biarpun demikian

kemungkinan juga bias menggunakan berbagai teknik yang lain seperti *roulette wheel* atau *tournament* [N.Srinivas and Kalyanmoy Deb 1994].

Individu yang ada pada *front* yang pertama mempunyai nilai fitness yang paling tinggi. Metode ini difokuskan untuk mencari daerah yang *non dominan* dan membantu untuk individu diatas daerahnya [Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan 2002]. Fitness *sharing* didalam algoritma *genetika*, teknik *sharing* tujuannya membuat formasi dan memperbaiki kestabilan dari sub populasi [Yau, Y. J.,; Teo, J.; and Anthony 2007].

2.2.6 Algoritma Genetika

Metode adaptif yang sering digunakan untuk memecahkan masalah dalam pencarian nilai dalam sebuah optimasi adalah algoritma *genetika*. Algoritma *genetika* berdasarkan proses *genetik* pada makhluk hidup, dengan proses berkembangnya generasi dalam sebuah populasi yang secara lambat laun berdasarkan seleksi alam dalam kemampuan bertahan hidup (*survive*). Teori evolusi inilah yang digunakan dengan metode algoritma *genetika* untuk mencari solusi dari berbagai macam permasalahan yang diciptakan oleh J.H Holland [16].

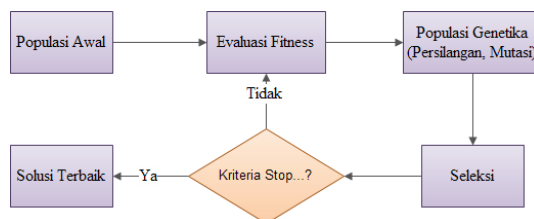
2.2.6.1 Pengertian Algoritma Genetika [17]

Algoritma ini ditemukan di Universitas Michigan, Amerika Serikat oleh John Holland (1975) melalui sebuah penelitian dan dipopulerkan oleh salah satu muridnya, David Goldberg (1989). Dimana mendefenisikan algoritma *genetic* ini sebagai metode algoritma pencarian berdasarkan pada mekanisme seleksi alam dan *genetik* alam. Algoritma *genetik* adalah algoritma yang berusaha menerapkan pemahaman mengenai evolusi alamiah pada tugas-tugas pemecahan-masalah (*problem solving*). Pendekatan yang diambil oleh algoritma ini adalah dengan menggabungkan secara acak berbagai pilihan solusi terbaik di dalam suatu kumpulan untuk mendapatkan generasi solusi terbaik berikutnya yaitu pada suatu kondisi yang memaksimalkan kecocokannya atau lazim disebut *fitness*. Generasi ini akan merepresentasikan perbaikan-perbaikan pada populasi awalnya. Dengan melakukan proses ini secara berulang, algoritma ini diharapkan dapat mensimulasikan proses evolusioner. Pada akhirnya, akan didapatkan solusi-solusi

yang paling tepat bagi permasalahan yang dihadapi. Untuk menggunakan algoritma *genetik*, solusi permasalahan direpresentasikan sebagai khromosom.

2.2.6.2 Struktur Umum Algoritma *Genetika*

Algoritma *genetik* memberikan suatu pilihan bagi penentuan nilai parameter dengan meniru cara reproduksi *genetik*, pembentukan kromosom baru serta seleksi alami seperti yang terjadi pada makhluk hidup. Algoritma *Genetik* secara umum dapat diilustrasikan dalam diagram alir berikut ini :



Gambar 2.9 Blok Diagram Algoritma *Genetika*

Golberg (1989) mengemukakan bahwa algoritma *genetik* mempunyai karakteristik-karakteristik yang perlu diketahui sehingga dapat terbedakan dari prosedur pencarian atau optimasi yang lain, yaitu:

1. Algoritma *genetik* dengan pengkodean dari himpunan solusi permasalahan berdasarkan parameter yang telah ditetapkan dan bukan parameter itu sendiri.
2. Algoritma *genetik* pencarian pada sebuah solusi dari sejumlah individu-individu yang merupakan solusi permasalahan bukan hanya dari sebuah individu.
3. Algoritma *genetik* informasi fungsi objektif (fitness), sebagai cara untuk mengevaluasi individu yang mempunyai solusi terbaik, bukan turunan dari suatu fungsi.
4. Algoritma *genetik* menggunakan aturan-aturan transisi peluang, bukan aturanaturan deterministik.

Variabel dan parameter yang digunakan pada algoritma *genetik* adalah:

1. Fungsi fitness (fungsi tujuan) yang dimiliki oleh masing-masing individu untuk menentukan tingkat kesesuaian individu tersebut dengan criteria yang ingin dicapai.
2. Populasi jumlah individu yang dilibatkan pada setiap generasi.
3. Probabilitas terjadinya persilangan (crossover) pada suatu generasi.
4. Probabilitas terjadinya mutasi pada setiap individu.
5. Jumlah generasi yang akan dibentuk yang menentukan lama penerapan algoritma *genetik*.

Secara umum struktur dari suatu algoritma *genetik* dapat mendefenisikan dengan langkah-langkah sebagai berikut:

1. Membangkitkan populasi awal

Populasi awal ini dibangkitkan secara random sehingga didapatkan solusi awal. Populasi itu sendiri terdiri atas sejumlah kromosom yang merepresentasikan solusi yang diinginkan.

2. Membentuk generasi baru

Untuk membentuk generasi baru, digunakan operator reproduksi/ seleksi, crossover dan mutasi. Proses ini dilakukan berulang-ulang sehingga didapatkan jumlah kromosom yang cukup untuk membentuk generasi baru dimana generasi baru ini merupakan representasi dari solusi baru. Generasi baru ini dikenal dengan istilah anak (offspring).

3. Evaluasi solusi

Pada tiap generasi, kromosom akan melalui proses evaluasi dengan menggunakan alat ukur yang dinamakan fitness. Nilai fitness suatu kromosom menggambarkan kualitas kromosom dalam populasi tersebut. Proses ini akan mengevaluasi setiap populasi dengan menghitung nilai fitness setiap kromosom dan mengevaluasinya sampai terpenuhi kriteria berhenti. Bila kriteria berhenti belum terpenuhi maka akan dibentuk lagi generasi baru dengan mengulangi langkah 2. Beberapa kriteria berhenti sering digunakan antara lain: berhenti pada generasi tertentu, berhenti setelah dalam beberapa generasi berturut-turut didapatkan nilai fitness tertinggi tidak berubah, berhenti dalam n generasi tidak didapatkan nilai fitness yang lebih tinggi.

2.2.6.3 Pengkodean

Pengkodean adalah suatu teknik untuk menyatakan populasi awal sebagai calon solusi suatu masalah ke dalam suatu kromosom sebagai suatu kunci pokok persoalan ketika menggunakan algoritma *genetik*. Berdasarkan jenis symbol yang digunakan sebagai nilai suatu *gen*, metode pengkodean dapat diklasifikasikan sebagai berikut:

2.2.6.3.1 Pengkodean biner

Merupakan cara pengkodean yang paling umum digunakan karena adalah yang pertama kali digunakan dalam algoritma *genetik* oleh Holland. Keuntungan pengkodean ini adalah sederhana untuk diciptakan dan mudah dimanipulasi. Pengkodean biner memberikan banyak kemungkinan untuk kromosom walaupun dengan jumlah nilai-nilai yang mungkin terjadi pada suatu *gen* yang sedikit (0 dan 1). Di pihak lain, pengkodean biner sering tidak sesuai untuk banyak masalah dan kadang pengoreksian harus dilakukan setelah operasi crossover dan mutasi.

Tabel 2.5 Contoh Pengkodean Biner

Kromosom A	1 0 1 1 0 0 1 0 1 1 0 0 0 1 0 1 1 1 0 0 0 1
Kromosom B	1 1 0 1 1 1 1 0 0 1 1 0 0 0 0 1 1 0 0 0 0

2.2.6.3.2 Pengkodean Bilangan Riil atau Pengkodean Permutasi

Suatu pengkodean bilangan dalam bentuk riil. Masalah optimalisasi fungsi dan optimalisasi kendala lebih tepat jika diselesaikan dengan pengkodean bilangan riil karena struktur topologi ruang *genotif* untuk pengkodean bilangan riil identik dengan ruang fenotifnya, sehingga mudah membentuk operator *genetic* yang efektif dengan cara memakai teknik yang dapat digunakan yang berasal dari metode konvensional.

Tabel 2.6 Contoh Pengkodean Bilangan Riil atau Bilangan Permutasi

Kromosom A	1 5 3 2 6 4 7 9 8
Kromosom B	8 5 6 7 2 3 1 4 9

2.2.6.3.3 Pengkodean Nilai

Pengkodean ini menyatakan setiap kromosom berupa string nilai aslinya. Nilai tersebut dapat berupa angka, huruf, atau kombinasi keduanya, disesuaikan dengan kasusnya. Metode ini lebih efektif dan tampak alami karena sesuai kasus yang diselesaikan.

Tabel 2.7 Contoh Pengkodean Nilai

Kromosom A	1.2324 5.3243 0.4556 2.3293 2.4545
Kromosom B	ABDJEIFJDHDIERJFDLDFLFEGT
Kromosom C	(back), (back), (right), (forward), (left)

2.2.6.4 Fungsi Evaluasi Kebugaran (*fitness*)

Fungsi kebugaran menggambarkan hasil atau solusi yang telah dikodekan. Selama proses GA induk harus digunakan *crossover* dan mutase untuk memperoleh keturunan. Jika GA dirancang dengan baik populasi akan mengalami sebuah *konvergensi* dan mengarah pada solusi yang optimum. Pada permulaan optimasi, biasanya nilai kebugaran akan memiliki rentang yang lebar. Seiring dengan bertambahnya generasi beberapa kromosom mendominasi populasi dan mengakibatkan rentang nilai kebugaran makin kecil. Hal ini dapat mengakibatkan *konvergensi* dini. Dalam hal ini hanya proses mutase yang mampu menghasilkan kromosom yang relative baru dan merupakan cara untuk menghindari kromosom tertentu mendominasi populasi.

2.2.6.5 Operator Genetik

Algoritma *genetik* merupakan proses pencarian yang heuristik dan acak sehingga penekanan pemilihan operator yang digunakan sangat menentukan keberhasilan algoritma *genetik* dalam menemukan solusi optimum suatu masalah yang diberikan. Hal yang harus diperhatikan adalah menghindari terjadinya *konvergensi premature*, yaitu mencapai solusi optimum yang belum waktunya, dalam arti bahwa solusi yang diperoleh adalah hasil optimum lokal. Operator *genetik* yang digunakan setelah proses evaluasi tahap pertama membentuk populasi baru dari generasi sekarang. Operator-operator tersebut adalah operator seleksi, *crossover* dan mutasi.

2.2.6.5.1 Seleksi

Seleksi bertujuan memberikan kesempatan reproduksi yang lebih besar bagi anggota populasi yang paling fit. Langkah pertama dalam seleksi ini adalah pencarian nilai fitness. Masing-masing individu dalam suatu wadah seleksi akan menerima probabilitas reproduksi yang tergantung pada nilai objektif dirinya sendiri terhadap nilai objektif dari semua individu dalam wadah seleksi tersebut. Nilai fitness inilah yang nantinya akan digunakan pada tahap seleksi berikutnya (Kusumadewi, 2003).

Kemampuan algoritma *genetik* untuk memproduksi kromosom yang lebih baik secara progresif tergantung pada penekanan selektif (*selective pressure*) yang diterapkan ke populasi. Penekanan selektif dapat diterapkan dalam dua cara. Cara pertama adalah membuat lebih banyak kromosom anak yang dipelihara dalam populasi dan memilih hanya kromosom-kromosom terbaik bagi generasi berikut.

Walaupun orang tua dipilih secara acak, metode ini akan terus menghasilkan kromosom yang lebih baik berhubungan dengan penekanan selektif yang diterapkan pada individu anak tersebut. Cara lain menerapkan penekanan selektif adalah memilih orang tua yang lebih baik ketika membuat keturunan baru.

Dengan metode ini, hanya kromosom sebanyak yang dipelihara dalam populasi yang perlu dibuat bagi generasi berikutnya. Walaupun penekanan selektif tidak diterapkan ke level keturunan, metode ini akan terus menghasilkan kromosom yang lebih baik, karena adanya penekanan selektif yang diterapkan ke orangtua. Ada beberapa metode untuk memilih kromosom yang sering digunakan antara lain adalah seleksi roda rolet (*roulette wheel selection*), seleksi ranking (*rank selection*) dan seleksi turnamen (*tournament selection*). Metode seleksi yang sering digunakan yaitu turnamen dan mesin *roulette* [18]. Dan dalam penelitian ini, seleksi yang digunakan adalah seleksi turnamen. Berikut masing – masing penjelasannya :

1. Turnamen

Metode turnamen ini diawali dengan menetapkan suatu nilai random untuk individu - individu yang disebut nilai *tour*. Parameter yang digunakan pada metode ini adalah ukuran *tour* yang bernilai antara 2 sampai N (dimana N adalah jumlah individu dalam suatu populasi).

2. Mesin *roulette*

Metode mesin *roulette* ini paling banyak digunakan karena merupakan metode yang paling sederhana atau sering disebut dengan *stochastic sampling with replacement*. Alur metode mesin *roulette* ini adalah sebagai berikut:

1. Masing-masing individu dihitung nilai *fitness*-nya (f_i adalah nilai *fitness* ke- i , untuk i adalah individu ke-1 s/d ke- n).
2. Dihitung total *fitness* semua individu dalam satu populasi.
3. Dihitung probabilitas masing-masing individu dalam satu populasi.
4. Dari probabilitas tersebut, dibangkitkan nilai random untuk masing – masing individu misalnya nilai random angka 1 sampai 100.

Ditentukan individu yang terpilih dalam proses seleksi berdasarkan hasil pembangkitan bilangan random.

2.2.6.5.2 Crossover

Crossover (perkawinan silang) bertujuan menambah keanekaragamanstring dalam populasi dengan penyilangan antar-string yang diperoleh dari sebelumnya. Beberapa jenis crossover tersebut adalah:

1. Crossover 1-titik

Pada crossover dilakukan dengan memisahkan suatu string menjadi dua bagian dan selanjutnya salah satu bagian dipertukarkan dengan salah satu bagian dari string yang lain yang telah dipisahkan dengan cara yang sama. Proses yang demikian dinamakan operator crossover satu titik seperti diperlihatkan pada gambar dan tabel berikut :

Tabel 2.8 Tabel *Crossover* 1-Titik

Kromosom Orang Tua 1	1 1 0 0 1 0 1 1
Kromosom Orang Tua 1	1 1 0 1 1 1 1 1
Keturunan	1 1 0 0 1 1 1 1

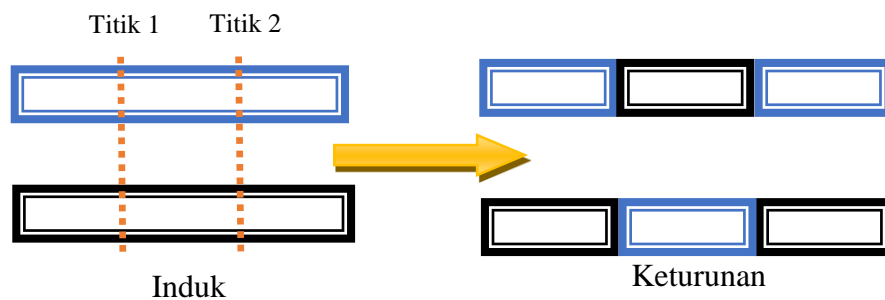
Gambar 2.10 Ilustrasi Operator Dengan Satu Titik Persilangan

2. Crossover 2-titik

Proses crossover ini dilakukan dengan memilih dua titik crossover. Kromosom keturunan kemudian dibentuk dengan barisan bit dari awal kromosom sampai titik crossover pertama disalin dari orangtua pertama, bagian dari titik crossover pertama dan kedua disalin dari orangtua kedua, kemudian selebihnya disalin dari orangtua pertama lagi.

Tabel 2.9 Tabel *Crossover* 2-Titik

Kromosom Orang Tua 1	1 1 0 0 1 0 1 1
Kromosom Orang Tua 2	1 1 0 1 1 1 1 1
Keturunan	1 1 0 1 1 1 1 1



Gambar 2.11 Ilustrasi Operator Dengan Dua Titik Persilangan

3. Crossover Seragam

Crossover seragam menghasilkan kromosom keturunan dengan menyalin bitbit secara acak dari kedua orangtuanya.

Tabel 2.10 Tabel *Crossover* Seragam

Kromosom Orang Tua 1	1 1 0 0 1 0 1 1
Kromosom Orang Tua 2	1 1 0 1 1 1 1 1
Keturunan	1 1 0 1 1 1 1 1

2.2.6.5.3 Mutasi

Mutasi merupakan proses mengubah nilai dari satu atau beberapa *gen*

dalam suatu kromosom. Operasi crossover yang dilakukan pada kromosom dengan tujuan untuk memperoleh kromosom-kromosom baru sebagai kandidat solusi pada generasi mendatang dengan fitness yang lebih baik, dan lama-kelamaan menuju solusi optimum yang diinginkan. Akan tetapi, untuk mencapai hal ini, penekanan selektif juga memegang peranan yang penting. Jika dalam proses pemilihan kromosom-kromosom cenderung pada kromosom yang memiliki fitness yang tinggi saja, konvergensi premature, yaitu mencapai solusi yang optimal lokal sangat mudah terjadi. Untuk menghindari konvergensi premature tersebut dan tetap menjaga perbedaan (diversity) kromosom-kromosom dalam populasi, selain melakukan penekanan selektif yang lebih efisien, operator mutasi juga dapat digunakan.

Proses mutasi dalam sistem biologi berlangsung dengan mengubah isi allele *gen* pada suatu locus dengan allele yang lain. Proses mutasi ini bersifat acak sehingga tidak selalu menjamin bahwa setelah proses mutasi akan diperoleh kromosom dengan fitness yang lebih baik. Operator mutasi merupakan operasi yang menyangkut satu kromosom tertentu.

2.2.6.5.4 Elitism

Selain kedua operator diatas (pindah silang dan mutase), masih ada satu lagi operasi yang dapat dilakukan dengan Algoritma *Genetika*, yaitu *elitism*. Operator ini bertujuan untuk menghilangkan kromosom terburuk dari satu generasi dan menggantikannya dengan kromosom terbaik dari generasi sebelumnya. Elitisme ini adalah proses pengopian satu atau lebih individu yang bernilai fitness tinggi agar tidak hilang selama evolusi. Ini dilakukan karena seleksi dilakukan secara acak, maka tidak ada jaminan bahwa suatu individu bernilai fitness tertinggi akan selalu terpilih. Jika terpilih mungkin individu tersebut akan rusak karena proses pindah silang (crossover). Prosedur ini digunakan pada Algoritma *Genetik* berjenis *generational replacement*.

2.2.6.5.5 Parameter Algoritma *Genetika*

Pengoperasian algoritma *genetik* dibutuhkan 4 parameter (Juniawati, 2003) yaitu :

1. Probabilitas Persilangan (Crossover Probability) $\rightarrow (P_c)$

Menunjukkan kemungkinan crossover terjadi antara 2 kromosom. Jika tidak terjadi crossover maka keturunannya akan sama persis dengan kromosom orangtua, tetapi tidak berarti generasi yang baru akan sama persis dengan generasi yang lama. Jika probabilitas crossover 100% maka semua keturunannya dihasilkan dari crossover. Crossover dilakukan dengan harapan bahwa kromosom yang baru akan lebih baik. Probabilitas ini digunakan untuk mengendalikan operator pindah silang, Dalam hal ini, dalam populasi terdapat **$P_c \times \text{POPSIZE}$** struktur (individu) yang melakukan pindah silang. Semakin besar nilai probabilitas pindah silang maka semakin cepat struktur baru diperkenalkan dalam populasi. Namun jika probabilitas pindah silang terlalu besar maka struktur dengan nilai obyektif yang baik dapat hilang dengan lebih cepat dari seleksi. Sebaliknya probabilitas pindah silang kecil maka akan menghalangi proses pencarian dalam GA. Menurut [Zbigniew Michalewics 1996] banyak aplikasi GA yang menggunakan nilai probabilitas pindah silang antara 0.65 – 1.

2. Probabilitas Mutasi (Mutation Probability) $\rightarrow (P_m)$

Menunjukkan kemungkinan mutasi terjadi pada *gen-gen* yang menyusun sebuah kromosom. Jika tidak terjadi mutasi maka keturunan yang dihasilkan setelah crossover tidak berubah. Jika terjadi mutasi bagian kromosom akan berubah. Jika probabilitas 100%, semua kromosom dimutasi. Jika probabilitasnya 0%, tidak ada yang mengalami mutasi. Mutasi digunakan untuk meningkatkan variasi populasi Probabilitas mutase ini digunakan untuk menentukan tingkat mutase yang terjadi, karena frekwensi terjadinya mutase tersebut menjadi **$P_m \times \text{POPSIZE} \times N$** , dimana N adalah panjang struktur atau *gen* dalam satu individu. Probabilitas yang rendah akan menyebabkan *gen – gen* yang berpotensi tidak dicoba. Dan sebaliknya tingkat mutase yang tinggi akan menyebabkan keturunan akan semakin mirip

dengan induknya. Dalam GA mutase menjalankan aturan penting yaitu :

- a. Mengganti *gen* yang hilang selama proses seleksi
- b. Menyediakan *gen – gen* yang tidak muncul pada saat inisialisasi alam populasi.

Banyak aplikasi GA yang menggunakan angka probabilitas antara 0.00 – 0.1

3. Jumlah Individu

Menunjukkan jumlah kromosom yang terdapat dalam populasi (dalam satu generasi). Jika hanya sedikit kromosom dalam populasi maka algoritma *genetic* akan mempunyai sedikit variasi kemungkinan untuk melakukan crossover antara orangtua karena hanya sebagian kecil dari search space yang dipakai. Sebaliknya jika terlalu banyak maka algoritma *genetik* akan berjalan lambat.

4. Jumlah Populasi

Menentukan jumlah populasi atau banyaknya generasi yang dihasilkan, digunakan sebagai batas akhir proses seleksi, persilangan dan mutasi.

5. Panjang Kromosom

Panjang kromosom berbeda – beda sesuai dengan model permasalahan. Titik solusi dari ruang permasalahan dikodekan dalam bentuk kromosom / string yang terdiri dari komponen *genetic* terkecil yaitu *gen*. Pengkodean dapat memakai bilangan seperti string biner, integer, floating point dan abjad. Mekanisme secara umum dari GA digambarkan sebagai proses kerjanya diawali dengan inisialisasi satu rangkaian nilai random yang disebut dengan populasi. Setiap individu dalam satu populasi dinamakan kromosom. Setiap individu dievaluasi dengan menggunakan nilai kebugaran atau *fitness*. Untuk menghasilkan generasi selanjutnya sebagai individu baru yang disebut dengan *offspring*, dibentuk melalui persilangan dua individu dengan menggunakan operator pindah silang (*crossover*) dan memodifikasi sebuah kromosom atau disebut juga dengan mutasi.

2.2.7 Non-dominated Sorting Genetic Algorithm (NSGA-II)

Non-dominated sorting GA(NSGA) diperkenalkan oleh Srinivas dan Deb pada tahun 1994 [N. Srinivas and Kalyanmoy Deb 1994]. NSGA [19] adalah algoritma non-dominasi populer berbasis *genetik* untuk optimasi multi obyektif. Ini adalah algoritma yang sangat efektif tetapi telah umum dikritik karena kompleksitas komputasi, kurangnya elitisme dan untuk memilih nilai parameter optimal untuk parameter berbagi σ_{share} . Sebuah versi modifikasi, NSGA- II [15] dikembangkan, yang memiliki algoritma sorting yang lebih baik, menggabungkan elitisme dan tidak ada parameter berbagi perlu dipilih secara prioritas. Walaupun telah disebutkan diawal ada beberapa nilai *critism* dari NSGA.

Populasi diinisialisasi seperti biasa. Setelah penduduk di diinisialisasi populasi diurutkan berdasarkan non-dominasi dalam setiap depan. Makhluk depan pertama yang sama sekali non-dominan dalam populasi saat ini dan makhluk depan kedua didominasi oleh individu dalam front pertama saja dan depan berjalan seterusnya. Masing Masing individu dalam setiap depan ditetapkan peringkat (fitness) nilai atau berdasarkan depan yang mereka miliki. Individu di depan pertama diberi nilai fitness dari 1 dan individu dalam kedua ditetapkan nilai fitness sebagai 2 dan seterusnya.

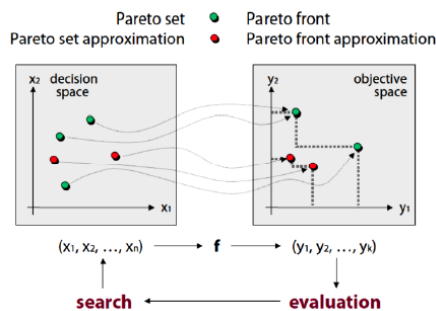
Selain nilai fitness parameter baru yang disebut *crowding distance* adalah menghitung setiap individu. *Crowding distance* adalah ukuran dari seberapa dekat individu dengan tetangganya. Besar rata-rata *crowding distance* akan menghasilkan keragaman yang lebih baik dalam populasi. Orang tua yang dipilih dari populasi dengan menggunakan turnamen seleksi biner berdasarkan pangkat dan *crowding distance*.

Seorang individu yang dipilih dalam peringkat ini lebih rendah dari yang lain atau jika *crowding distance* lebih besar dari yang lain. Yang dipilih adalah populasi yang menghasilkan keturunan dari crossover dan operator mutasi, yang akan dibahas secara rinci pada bagian selanjutnya. Populasi dengan populasi saat ini dan keturunan saat ini diurutkan lagi berdasarkan nondominasi dan hanya yang terbaik N individu yang dipilih, di mana N adalah ukuran populasi. Pemilihan ini didasarkan pada pangkat dan pada *crowding distance* di bagian depan sebelumnya.

Algoritma *Genetika* yang telah dijelaskan pada sub bab sebelumnya

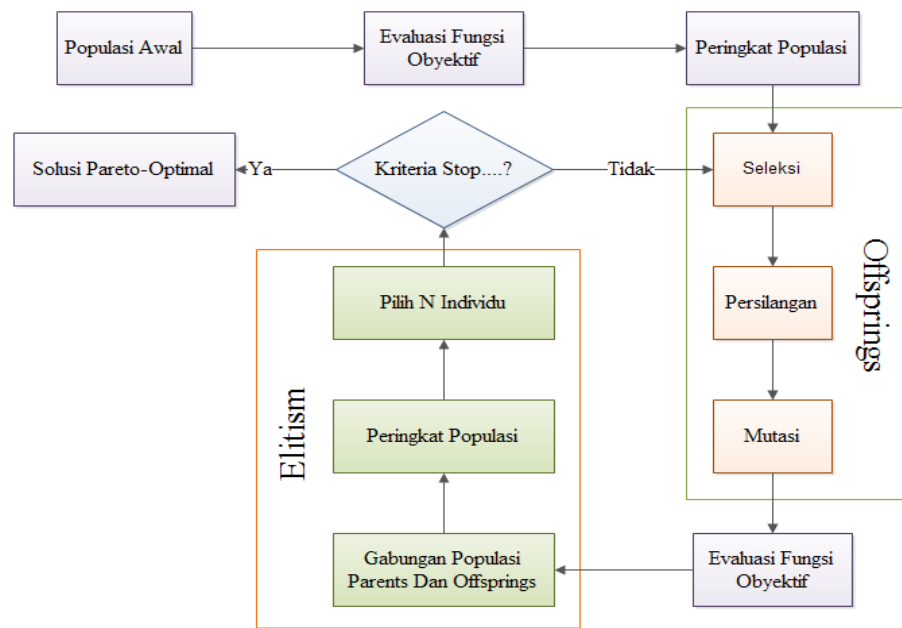
merupakan metode optimasi dengan obyektif tunggal. Metode tersebut digunakan untuk masalah optimasi skalar karena fungsi obyektif selalu mencapai nilai optimal global tunggal atau skalar. Untuk multi obyektif, fungsi multi-obyektif membentuk vektor sehingga biasa disebut dengan optimasi vektor [20].

Permasalahan optimasi multi obyektif tidak seperti permasalahan optimasi obyektif tunggal, tidak membutuhkan solusi optimal yang meminimalisasi semua fungsi multi obyektif secara simultan. Seringkali terjadi konflik antara obyektif satu dengan yang lain dan parameter optimal dari obyektif biasanya tidak mengarahkan kepada optimalitas obyektif lainnya (terkadang membuat lebih buruk). Sebagai contoh, sebuah keluarga yang menginginkan pelayanan kelas satu saat liburannya, namun menginginkan harga yang murah. Pelayanan tingkat tinggi (satu obyektif) akan sangat memakan biaya, namun ini menyebabkan konflik dengan obyektif lainnya (untuk meminimalisir biaya) [10].



Gambar 2.12 Ruang Keputusan dan Obyektif (*Decision and Objective Space*) [10]

Langkah utama pada NSGA-II adalah bahwa n *offspring* (S_t) diciptakan dari n *parent* (P_t) menggunakan algoritma genetika standar. Populasi keseluruhan (W_t) berukuran $2n$ dibentuk dengan menggabungkan S_t dan P_t ($W_t = P_t \cup S_t$). Kemudian sorting nondominasi diterapkan kepada W_t . Kemudian populasi baru akan terisi, satu kali dalam satu waktu, oleh solusi nondominasi yang berbeda. Karena populasi dari W_t adalah $2n$, hanya separuh dari jumlah tersebut yang akan menjadi populasi baru, memilih solusi nondominasi dari Pareto front dengan keragaman tinggi dan membuang solusi yang tersisa.



Gambar 2.13 Blok Diagram NSGA-II [10]

2.2.7.1 Inisialisasi Populasi

Inisialisasi populasi ini merupakan langkah awal algoritma NSGA-II. Populasi diinisialisasi berdasarkan range dari permasalahan atau parameter-parameter yang lain jika itu ada.

2.2.7.2 Non-Dominated Sort

Inisialisasi populasi yang di-*sort* berdasarkan non itness algoritma digambarkan dibawah ini :

Algoritma 2.1 Algoritma Non-Dominated Sort

1. Individu p pada populasi P akan diolah sebagai berikut :
 - a. Buat kumpulan $S_p = \{ \}$ berisi nama kumpulan individu yang itness dengan p
 - b. Buat counter $a_p = 0$ sebagai penghitung jumlah dari individu yang itness di $S_p = \{ \}$
 - c. Misal untuk individu q di P
 - 1) Jika p didominasi q maka
 Tambahkan q ke himpunan S_p i.e $S_p = S_p \cup \{q\}$
 - 2) Selain itu jika q itness di p then
 Increment itness hitung p i.e $n_p = n_p + 1$

- d. if $n_p = 0$, tidak ada individu yang fitness di p then p adalah *front* pertama
 $p_{rank} = 1$ update *front* pertama dengan menambah p ke *front* 1 $F_1 = F_1 \cup \{p\}$
2. Kemudian akan dibawa untuk semua individu didalam populasi P
3. Inisialisasi penghitung *front* ke 1 $i = 1$
4. Ketika i^{th} *front* tidak kosong $F_i \neq \emptyset$
 - a. $Q = \{\}$ untuk menyimpan individu dari $(i + 1)^{th}$ *front*.
 - b. For each individu q di S_p (S_p adalah kumpulan dari individu yang didominasi p)
 - 1) $N_q = n_q - 1$, *decrement* penghitungan dominasi untuk individu q .
 - 2) if $n_q = 0$ then tidak ada individu dalam subsequent *front* akan didominasi q . kemudian $q_{rank} = i + 1$. Update himpunan Q dengan individu q $Q = Q \cup q$
 - c. Increment *front counter* dengan 1
 - d. Sekarang himpunan Q adalah *front* dan $F_i = Q$

2.2.7.3 Crowding Distance

Crowding distance dapat dihitung sebagai berikut :

Algoritma 2.2 Algoritma Crowding Distance

1. For each front F_i , n adalah jumlah dari individu
 - a. Inisialisasi jarak menjadi 0 untuk semua individu $F_i(d_j)$ dimana j berelasi ke individu j^{th} di front F_i
 - b. For each fungsi objektif m
2. Sort individu di front F_i berdasarkan objektif m
 $I = \text{sort}(F_i, m)$
3. Menentukan jarak ke nilai batas untuk masing – masing individu F_i $I(d_1) = \sim$ dan $I(d_n) = \sim$
4. For $k = 2$ to $(n-1)$
5. $I(k).m$ adalah nilai dari M^{th} fungsi objektif dari individu k^{th} di I

Ide dasar crowding distance adalah menemukan jarak Euclidian diantara masing – masing individu di sebuah front berdasarkan nilai m objektifnya didalam dimensi ruang m jarak pada individu di pilih ketika jarak itu ditentukan.

2.2.7.4 Selection

Seleksi dilakukan menggunakan *Crowed-comparison-operator* (\prec_n). Perbandingan dilakukan berdasarkan non domination P_{rank} dan *crowding distance*. Algoritmanya sebagai berikut :

Algoritma 2.3 Algoritma Selection

1. Non-domination rank P_{rank} individu di front F_i akan mempunyai rank sebagai $P_{rank} = i$
2. Crowding distance $F_i(d_j)$
 - a. $P \prec_n q$ if
 - 1) $P_{rank} < q_{rank}$
 - 2) Atau if p dan q mempunyai front F_i Then $F_i(d_p) > F_i(d_q)$

2.2.7.5 Genetic Operator

Kode GA menggunakan *Simulated Binary Crossover* (SBX). Untuk genetic operator ini terdiri dari *Simulated Binary Crossover* dan *Polynomial Mutation*.

2.2.7.6 Recombination dan Seleksi

Populasi *Offspring* digabungkan dengan generasi populasi yang terbaru. Seleksi membentuk kumpulan individu untuk generasi selanjutnya dan dipilih yang terbaik untuk ditambahkan pada populasi. Populasi dikelompokkan berdasarkan non dominasi. Generasi baru diisi oleh masing – masing *front* hingga ukuran populasi melampaui ukuran populasi baru. Jika penambahan semua individu di *front* F_j populasinya melebihi N maka dinividu di F_j dipilih berdasarkan *crowding distancenya* secara menurun sampai ukuran populasinya N dan prosesnya diulang untuk membangkitkan generasi berikutnya.

Halaman ini sengaja dikosongkan

BAB III

METODE PENELITIAN

Game space shooter adalah *game* dengan *genre* FPS yang memiliki tujuan menghancurkan musuh dan menghindari serangan musuh. Dalam *game space shooter* terdapat NPC yang memiliki perilaku sulit atau mudah di hadapi oleh *player*, sehingga *game* menjadi membosankan karena NPC memiliki perilaku statis.

Untuk memaksimalkan perilaku NPC dalam *game*, peneliti mencoba memberikan kemampuan adaptasi terhadap perilaku dari *player* untuk dapat mengevolusi parameter yang dimiliki oleh NPC, sehingga *game space shooter* menjadi lebih menarik dan menyenangkan, karena NPC dapat beradaptasi dengan perilaku *player* dengan merubah *health* dan *speed* nya sesuai dengan perilaku dari *player*.

Bab ini menjelaskan tahapan-tahapan yang dilakukan pada penelitian untuk mencapai tujuan tersebut. Dan hasil dari penelitian ini diterapkan pada evolusi dinamis NPC untuk menentukan evolusi yang optimal ketika dihadapkan dengan perilaku *player* yang dinamis.

3.1 Lingkup Penelitian

Tujuan penelitian ini adalah membangun suatu model *game* yang memiliki kondisi dimana NPC dapat meningkatkan kemampuannya seiring *game* sedang dimainkan agar dapat memberikan perlawanan yang sepadan kepada *player* sehingga terciptanya keseimbangan di dalam *game*. Terdapat 4 jenis senjata yang dimiliki oleh NPC yaitu senjata *gun*, *missile*, *rudal*, dan *suicide* yang digunakan untuk menghancurkan pesawat *player*.

Selama permainan berlangsung, jika perilaku *player* mudah dihadapi NPC maka NPC menurunkan *speed* dan *health* serta merubah senjatanya sehingga mudah dihadapi oleh *player* pada level berikutnya. Namun jika perilaku *player* terlalu sulit dihadapi oleh NPC, maka NPC dapat merubah *speed* dan *health* menjadi lebih tinggi agar sesuai dengan kemampuan dari *player* pada level berikutnya. Pemberian ruang lingkup penelitian ditujukan agar penelitian lebih terarah.



Gambar 3.1. Diagram Venn Lingkup Penelitian

Berdasarkan ketiga topik penelitian yang diambil, dikerucutkan menjadi suatu bagian spesifik dari topik-topik tersebut dikarenakan besarnya ruang lingkup dari topik penelitian. Topik *Game Behavior* yang menjadi permasalahan pada penelitian ini, diambil sub-genre FPS (*First person shooter*) untuk dijadikan bahan penelitian yang lebih difokuskan kepada optimasi evolusi dinamis NPC (*health* dan *speed* dari *Non-Player Character*).

3.2 Test-bed Penelitian

Test-bed yang digunakan dalam penelitian ini adalah *genre game* FPS yang dispesifikan pada *game space shooter*. *Test-bed* dibutuhkan untuk mengimplementasikan metode yang diajukan dan untuk melakukan percobaan sehingga pendekatan yang dikembangkan dapat dievaluasi. Sub-bab ini memberikan detail bagaimana *game* dibuat dan bagaimana *game* ini dapat mendukung penelitian. Sub-bab ini didiskripsikan menjadi beberapa bagian meliputi deskripsi dari *game*, desain 3D *game*, dan desain evolusi dinamis dari NPC.

3.2.1 Deskripsi Game

Space Shooter merupakan sebuah *game* bergenre FPS (*First Person Shooter*) dimana latar belakang *game* ini adalah pesawat luar angkasa yang sedang menjelajahi antariksa untuk menemukan planet baru yang dapat disinggahi oleh manusia bumi. Namun ditengah perjalanan *player* bertemu dengan musuh yang berusaha menghancurkan pesawat *player*.

Player harus dapat menghancurkan musuh sebanyak-banyaknya sebagai rintangan untuk mencapai garis akhir untuk dapat bertarung melawan pesawat bos yang berada pada akhir perjalanan dalam setiap level. Keberadaan bos dibuat sebagai rintangan terakhir untuk dapat naik kelevel selanjutnya. Jika *health* dari *player* mencapai 0 dan pesawat *player* hancur, maka *player* harus memulai game dari awal perjalanan. Namun jika *player* dapat mengalahkan bos penjaga pada level yang dimainkan, maka *player* berhak melanjutkan ke level selanjutnya. Selama permainan berlangsung, *player* menghadapi musuh dengan senjata *Gun*, *missile*, *rudal*, *suicide* serta perilaku musuh yang berbeda-beda.

Dalam pesawat musuh terdapat item yang dapat digunakan untuk mengaktifkan *shield* untuk melindungi pesawat *player* dari segala perilaku serangan musuh, item untuk menambah *hit point player*, dan item untuk menambah satu *rudal* setiap kali *player* dapat mengambil item yang terjatuh dari pesawat musuh. Agar *player* mendapatkan item yang terjatuh dari pesawat musuh, *player* harus menghancurkan pesawat musuh sehingga item yang terdapat pada pesawat musuh terjatuh dan dapat diambil oleh *player*.

Selama *game* berlangsung, jika *player* menghancurkan pesawat musuh, maka *player* mendapatkan score sebanyak 10. Dan jika *player* terkena serangan musuh, maka *health player* berkurang sesuai dengan senjata yang digunakan oleh musuh untuk menghadapi *player*.

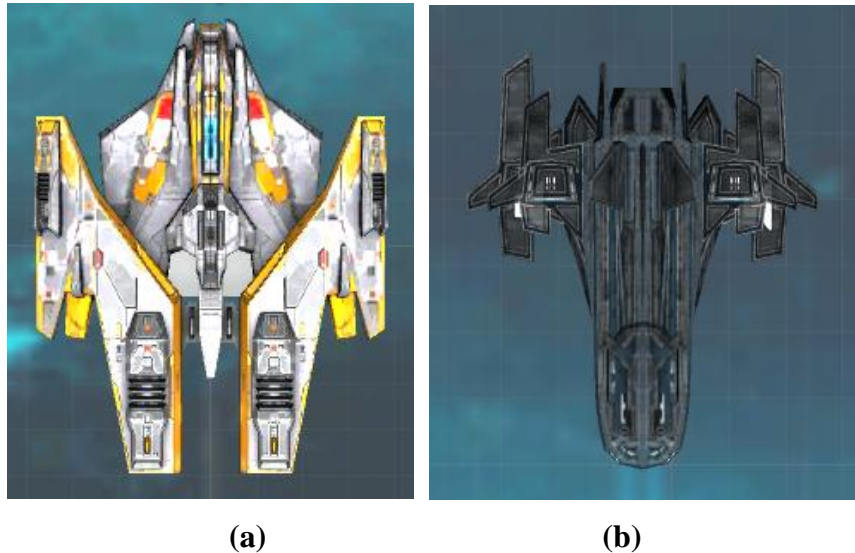
3.2.2 Desain 3D Game

Game adalah media yang memberikan sensasi *fun* terhadap suatu pengalaman tertentu kepada *player*. Sehingga perlu dibuat desain yang sesuai dengan storyboard dari *game* tersebut. Desain merupakan bagian penting dalam elemen estetika untuk pembuatan *game*, sehingga *player* tidak mudah bosan jika melihat tampilan visual dari *game*.

Elemen estetika tidak hanya menyangkut tampilan visual. Ada beberapa elemen lain yang terkait dengan elemen estetika. Misalnya *background music*, *sound effect*, *user interface* yang fungsional, serta *user-friendly* dan lain sebagainya. Dalam pembuatan *game*, *developer game* harus menyatukan elemen-elemen

penting tersebut menjadi estetika yang menarik dan fungsional, sehingga tercipta *game* yang *fun* untuk dimainkan.

Dalam penelitian ini digunakan model 3D untuk pembuatan pesawat tempur *player*, pesawat tempur musuh dan *weapon*. Seperti yang terlihat pada gambar 3.2.



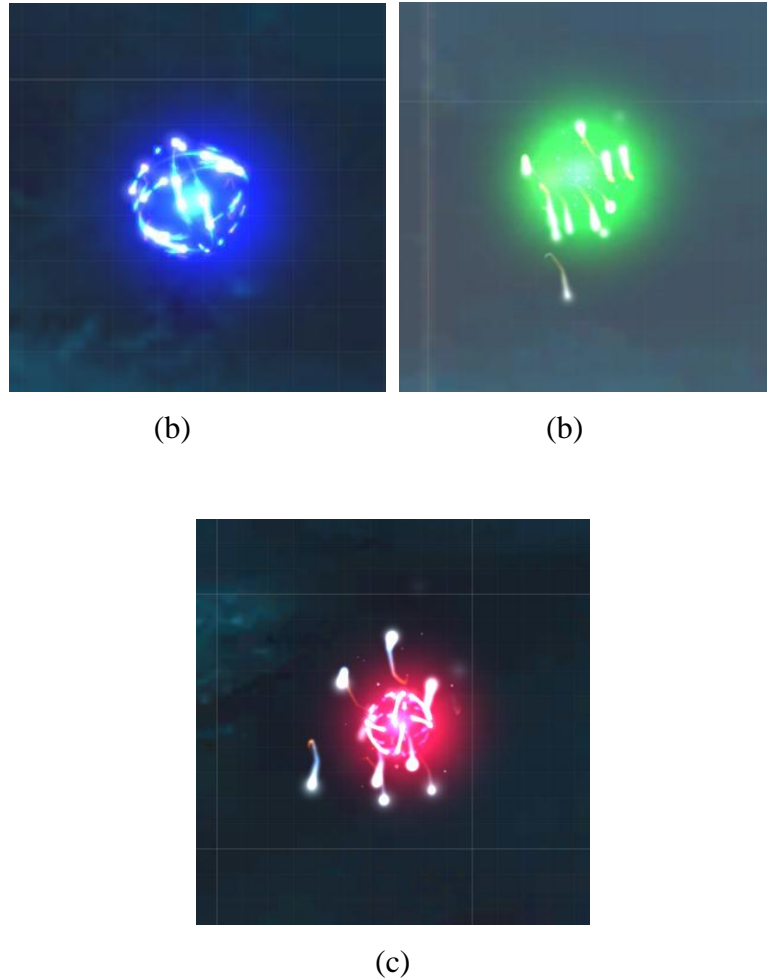
Gambar 3.2 Desain 3D Pesawat Tempur (a) pesawat *Player* (b) pesawat Musuh (NPC)

Selain desain 3D pesawat tempur *player* dan NPC terdapat pula desain item futuristic untuk memberikan efek visual yang sepadan, sehingga dapat memberikan nilai estetika kecanggihan teknologi dalam *game*. Terdapat tiga item yang disiapkan peneliti untuk membantu *player* dalam bermain. Untuk mendapatkan item tersebut *player* harus menghancurkan pesawat musuh dan mengambil item yang terjatuh dari pesawat musuh.

Untuk dapat membedakan macam-macam item digunakan tiga warna yang mencolok dan mudah di ingat, serta masing-masing dari item memiliki animasi dan *lightning* yang berbeda-beda agar memudahkan untuk dilihat oleh *player*, item dengan warna hijau adalah item untuk menambah *health player* sejumlah 20 HP(*health*), item dengan warna biru adalah item untuk menambah satu *weapon* (rudal) dalam stok senjata rudal milik pesawat *player*, dan item dengan warna merah adalah item untuk mengaktifkan *shield player* selama sepuluh detik yang dapat menahan semua jenis serangan dari pesawat NPC.

Ketika *player* berhasil menghancurkan pesawat musuh yang kebetulan

terdapat item didalamnya, maka item tersebut akan jatuh dan player hanya memiliki beberapa detik untuk mendapatkan item yang terjatuh dari pesawat musuh sebelum item menghilang atau terlewat sebelum dapat diambil oleh *player*. Sehingga menyebabkan pesawat *player* tidak mendapatkan item tersebut



Gambar 3.3 Desain Item Dengan Efek *Lightning* (a) Item Untuk Menambah satu Rudal Dalam Stock senjata (b) Item Untuk Menambahkan *health player* (c) Item Untuk Mengaktifkan *Shield Player*

Kondisi ketika *player* berhasil mendapatkan item *shield* yang dijatuhkan oleh musuh ketika pesawat musuh berhasil dihancurkan oleh *player* dapat dilihat pada gambar 3.4. Dalam keadaan *shield* aktif, *health player* tidak akan berkurang meskipun tertembak ataupun menabrak pesawat musuh. Namun *shield* hanya dapat aktif beberapa detik ketika item di dapatkan oleh *player*.



Gambar 3.4. Kondisi Ketika *Player* Mendapatkan Item *Shield*


3.2.3 Desain Senjata pada NPC

Sub-bab ini membahas tentang desain senjata dan armor yang dimiliki oleh player untuk dapat mengembangkan *game* menjadi semakin menarik seiring waktu dan level.

1. Senjata *gun* menyerang jarak dekat maupun jarak jauh, namun hanya memiliki *damage* yang kecil terhadap *health* musuh
2. Senjata *Missile* menyerang dalam jarak jauh maupun dekat, namun memiliki kelemahan mudah dihindari oleh musuh, karena *Missile* memiliki *speed* lebih rendah dari pada *gun*. Namun *missile* memiliki *size damage* yang lebih besar dari *gun*.
3. Senjata Rudal menyerang jarak jauh maupun jarak dekat, namun memiliki batas terhadap jumlah maupun jarak dari sensor rudal. Jika musuh ada diluar jangkauan radar. Maka rudal tidak dapat mengenai sasaran dengan tepat. *Damage* dari rudal setara dengan *damage* dari *missile*, namun memiliki *speed* setara dengan *gun*.
4. Perilaku serangan *Suicide* atau serangan buruh diri adalah serangan terkuat karena memiliki *damage attack* NPC paling besar. Dengan menabrakkan pesawat musuh kepada pesawat *player*. *Speed* yang dimiliki oleh pesawat *suicide* juga setara dengan senjata rudal dan *gun*.
5. Item *Armor* adalah item yang dapat menjadikan pesawat player kebal terhadap senjata maupun serangan *suicide* musuh, namun *health* musuh langsung habis jika menabrak pesawat *player* dalam keadaan *shield* aktif.

Tabel 3.1 Jenis – Jenis Senjata dan *Armor* Dalam Permainan

Gambar	Nama	Perilaku
	Gun	Menyerang Jarak Jauh dan Dekat Mengurangi Sedikit <i>Health</i> Speed Cepat
	<i>Missile</i>	Menyerang Jarak Jauh Dan Dekat Mengurangi Sedang <i>Health</i> Mudah Dihindari Oleh <i>Player</i> Karena Terlihat Oleh <i>Player</i> Speed Sedang
	Rudal	Menyerang Jarak Jauh Dan Dekat Mengurangi Banyak <i>Health</i> Musuh Harus Ada Dalam Jarak Radar Speed Sedang
	<i>Armor</i>	Menahan Semua Serangan Dari Musuh Tidak Mengurangi Speed dari Pesawat <i>Player</i> Dapat Digunakan Sebagai <i>Attack</i> Kepada Musuh

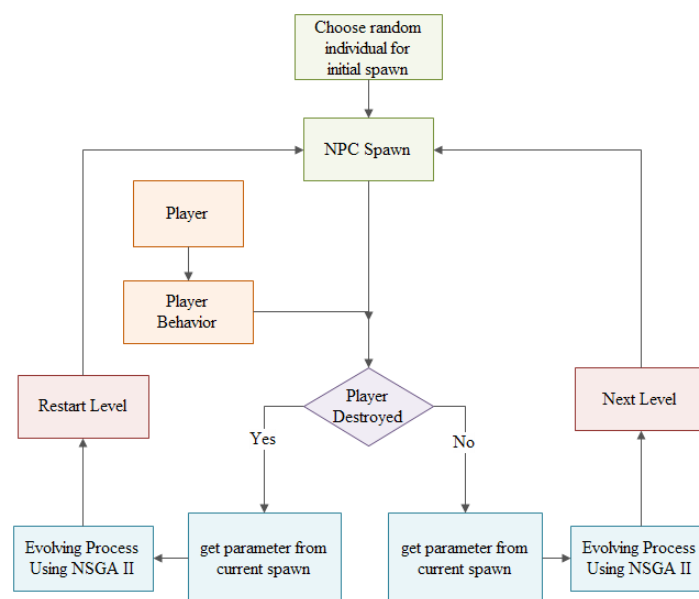
	<p><i>Suicide</i></p>	<p>Serangan Paling Kuat Dan Mematikan, Karena Menggunakan Seluruh Badan Pesawat Sebagai Senjata. Efek Benturan Dapat Menghilangkan Seluruh <i>Health</i>.</p>
---	-----------------------	---

Perubahan jenis senjata berlangsung ketika *game* telah berakhir dalam satu level. Nilai *health* dari *player*, *score* dari *player* serta *NPC missed* dan *damage player* dihitung untuk merubah jenis senjata yang digunakan NPC ketika *game* akan kembali dimulai (*next level* maupun *restart level*).

3.3 Tahapan Pembuatan Sistem

Penelitian ini difokuskan pada optimasi perilaku evolusi dinamis pada NPC (terfokus pada *speed* dan *health* dari NPC). Untuk mengoptimasi perilaku evolusi dinamis NPC harus melewati berbagai langkah atau tahapan, mulai dari membuat *game space shooter* beserta item, rancangan karakteristik *player*, desain evolusi dinamis NPC, optimasi evolusi dinamis NPC yang *multi-objective*.

Evolusi dinamis pada NPC dapat berubah sesuai dengan tahapan system yang telah dibuat. Agar tidak terjadi kesalahan dalam tahap uji coba.

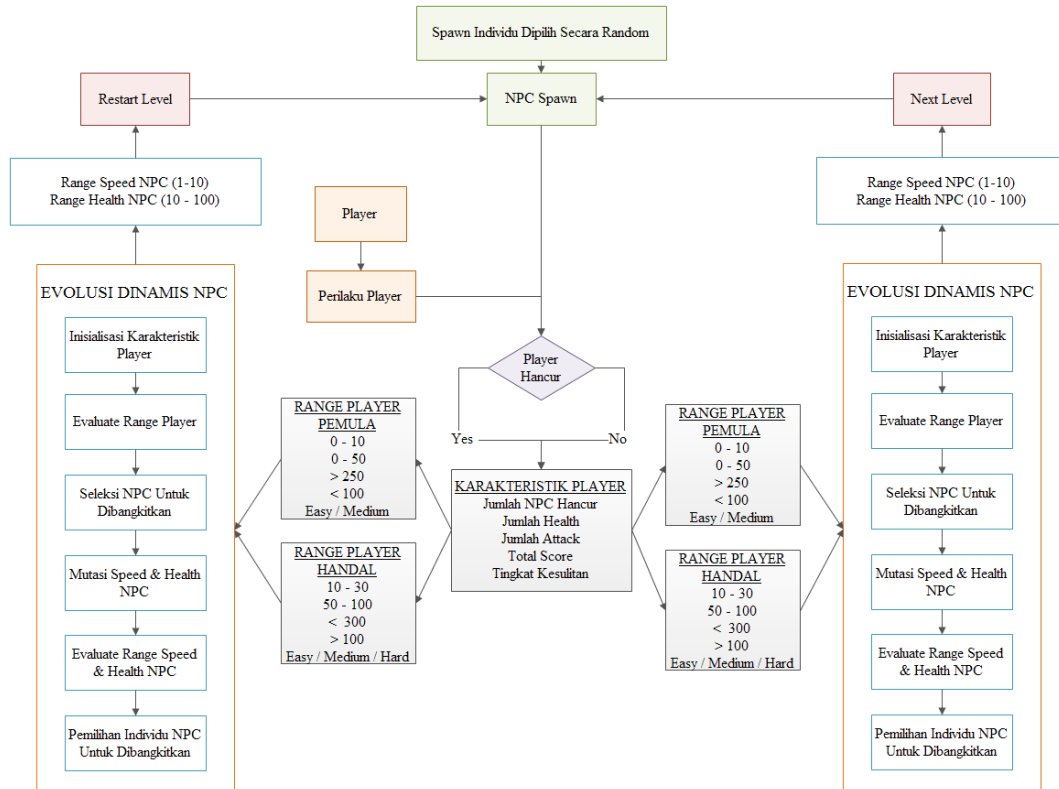


Gambar 3.5. Diagram Permainan

3.3.1 Evolusi Dinamis Dalam Permainan

Simulasi dalam permainan difokuskan pada optimasi *multi-objective* perilaku evolusi dinamis pada NPC (terfokus pada *speed* dan *health* dari NPC). Untuk mengoptimasi perilaku evolusi dinamis NPC harus melewati berbagai langkah atau tahapan, mulai dari membuat *game space shooter* beserta item, desain evolusi dinamis NPC, optimasi evolusi dinamis NPC yang *multi-objective*.

Evolusi dinamis pada NPC dapat berubah sesuai dengan tahapan system yang telah dibuat. Agar tidak terjadi kesalahan dalam tahap uji coba.



Gambar 3.6. Diagram Evolusi Dalam Permainan

Pada gambar 3.6 terlihat proses pengambilan data dari perilaku dan karakteristik *player*, yang selanjutnya dihitung dan diproses untuk menentukan pilihan evolusi dinamis NPC. Selanjutnya NPC menentukan range yang sesuai dengan perilaku dan karakteristik dari *player* pada level sebelumnya untuk dapat meningkatkan evolusi dinamis *health* dan *speed* dari NPC untuk level berikutnya dalam permainan.

3.3.2 Karakteristik *Player*

Sebelum simulasi dilakukan, maka ditentukan karakteristik *player* sebagai

rancangan penelitian ketika uji coba dilakukan, sehingga dapat diperoleh data akurat dari perilaku *player*. Karakteristik tersebut di bagi menjadi dua kategori, yaitu karakteristik *player* handal dan karakteristik *player* pemula, beserta dengan range nilai untuk membedakan data dari *player* handal dan *player* pemula, sehingga dapat diperoleh data akurat untuk uji coba evolusi dinamis NPC berdasarkan karakteristik *player*.

Karakteristik dari *player* pemula yang digunakan sebagai data evolusi dinamis NPC adalah jumlah NPC yang dapat dihancurkan oleh *player* berjumlah sedikit, scor NPC selama simulasi dilakukan rendah, jumlah attack *player* untuk menghancurkan NPC banyak, sisa *health player* dalam satu level sedikit, perubahan tingkat kesulitan permainan lambat (selama permainan berlangsung mulai dari level 1 hingga level 5, *player* pemula tidak dapat mencapai tingkat kesulitan *hard*).

Tabel 3.2 Karakterisrik *Player* Pemula

No	Karakteristik Player Pemula	Range <i>player</i> Pemula
1	Jumlah NPC Hancur	0 - 10
2	Jumlah Health	0 - 50
3	Jumlah Attack	> 250
4	Total Scor	< 100
5	Tingkat Kesulitan	Easy / Medium

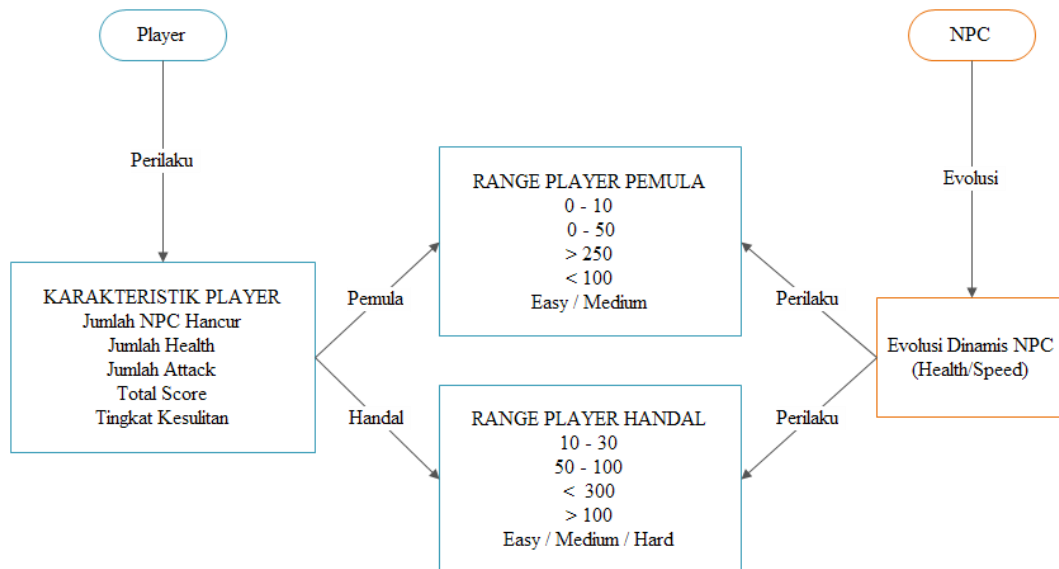
Sedangkan data karakteristik *player* handal yang digunakan sebagai data evolusi dinamis NPC adalah jumlah NPC yang dapat dihancurkan oleh *player* handal berjumlah banyak, scor NPC selama simulasi dilakukan tinggi, jumlah attack *player* untuk menghancurkan NPC sedikit, sisa *health player* dalam satu level banyak, perubahan tingkat kesulitan permainan dinamis (selama permainan berlangsung mulai dari level 1 hingga level 5, *player* handal dapat mencapai tingkat kesulitan dinamis (*easy*, *medium*, dan *hard*)).

Tabel 3.3 Karakterisrik *Player* Handal

No	Karakteristik Player Handal	Range <i>player</i> Handal
1	Jumlah NPC Hancur	10 - 30
2	Jumlah Health	50 - 100
3	Jumlah Attack	< 300

4	Total Scor	> 100
5	Tingkat Kesulitan	Easy / Medium / Hard

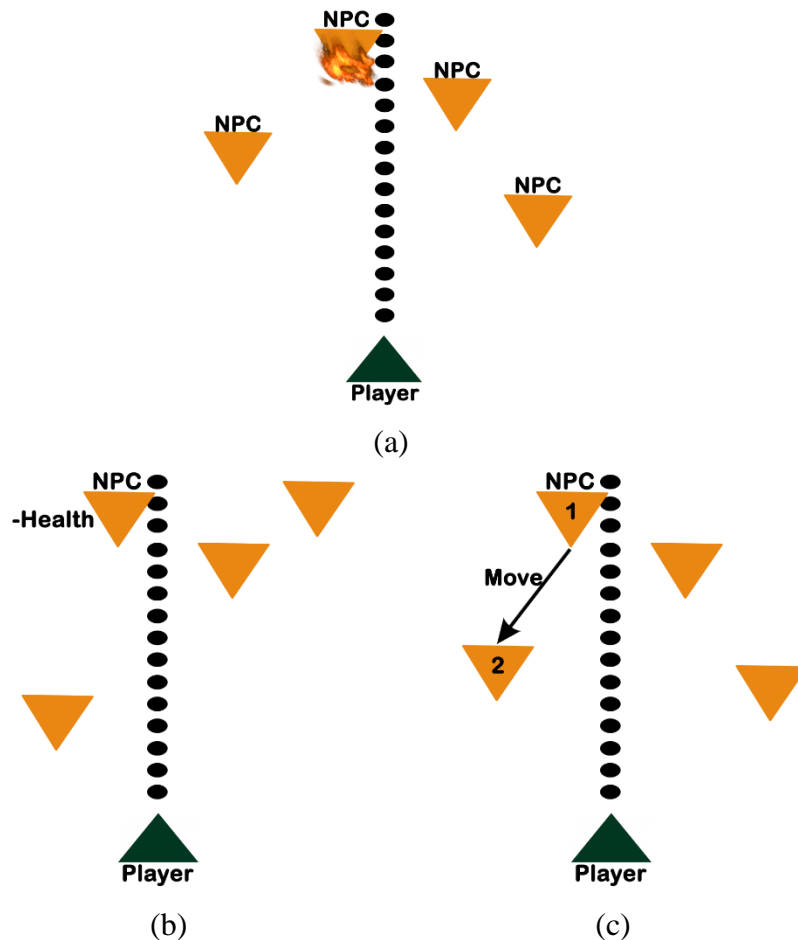
Penentuan karakteristik beserta range nilai dari *player* berguna untuk menentukan batasan kondisi dari *player* pemula dan *player* handal ketika menghadapi NPC. seperti yang terlihat pada gambar 3.7.



Gambar 3.7. Diagram Karakteristik *Player* Dalam Permainan

3.3.3 Perilaku NPC Terhadap *Player*

Selain perilaku dan karakteristik dari *player*. Dibuat juga rancangan evolusi dinamis NPC ketika menghadapi *player*. Sehingga selama permainan berlangsung NPC dapat meningkatkan kemampuannya berdasarkan perilaku dan karakteristik dari *player*. Pada penelitian ini terfokus pada evolusi dinamis *health* dan *speed* dari NPC dengan melihat perilaku dan karakteristik dari *player* pada level sebelumnya. Sehingga pada level berikutnya data dari perilaku dan karakteristik *player* diolah untuk menentukan tingkat evolusi *health* dan *speed* dari NPC. Sehingga NPC dapat menyesuaikan kemampuannya dengan karakteristik dan perilaku dari *player*.

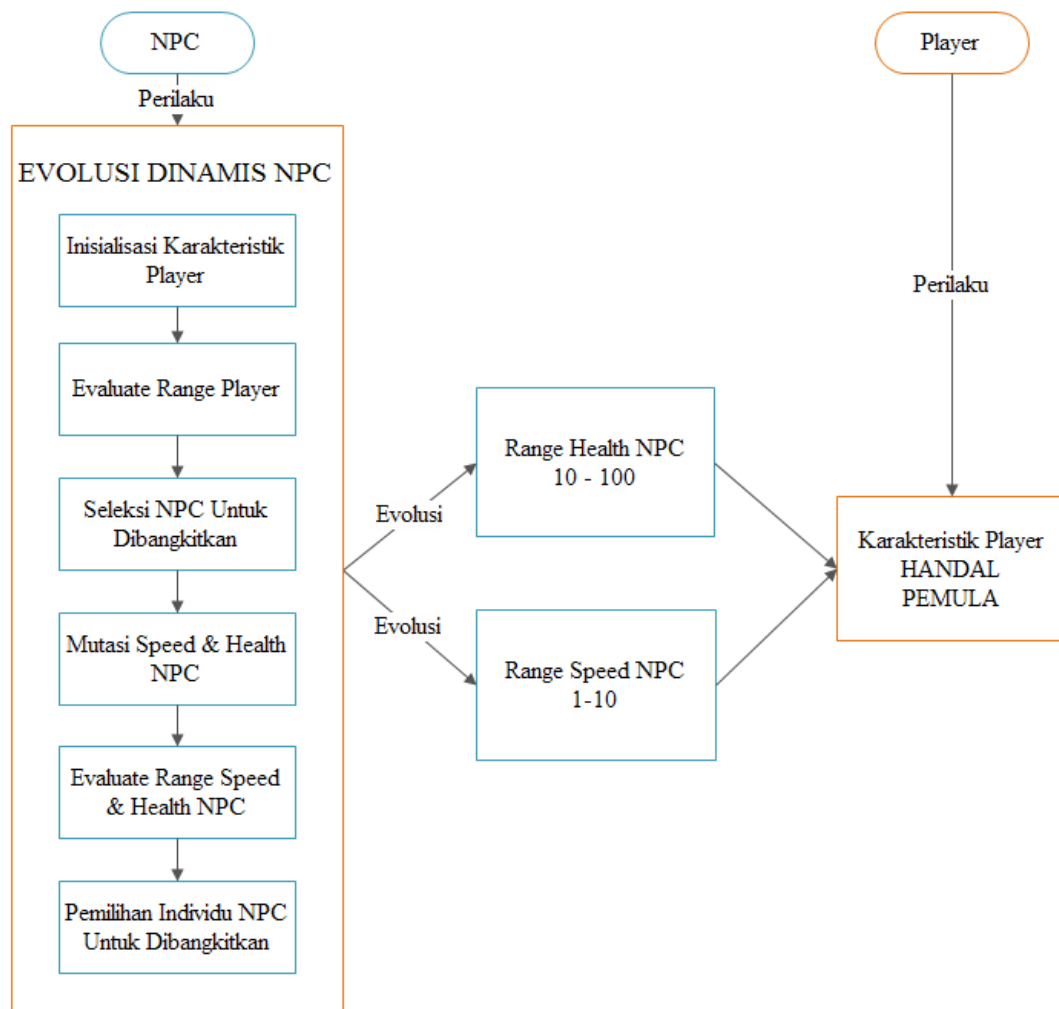


Gambar 3.8. Evolusi Dinamis NPC. (a) NPC terkena tembakan dari player. (b) NPC telah meningkatkan jumlah *health*. (c) NPC telah meningkatkan jumlah *speed* nya.

Rancangan dari evolusi dinamis *health* dan *speed* dari NPC seperti yang ditunjukkan pada gambar 3.8. pada gambar (a) adalah proses *player* sedang berusaha menghancurkan pesawat NPC. Pada saat permainan berlangsung, jika *player* berhasil menembak pesawat NPC maka *health* dari NPC akan berkurang sesuai dengan jumlah peluru yang mengenai pesawat NPC, jika *health* NPC mencapai 0 maka pesawat NPC akan hancur. Pada gambar (b) menunjukkan proses evolusi dinamis *health* NPC telah dilakukan sehingga jumlah *health* dari NPC telah meningkat, sehingga menyebabkan pesawat NPC semakin sulit untuk dihancurkan oleh *player*. Pada gambar (c) menunjukkan proses evolusi dinamis *speed* NPC telah dilakukan sehingga *speed* dari NPC telah bertambah, sehingga menyebabkan tembakan dari pesawat *player* semakin sulit untuk mengenai NPC.

Asumsi yang dibangun dalam simulasi pada permainan adalah dengan

menentukan range evolusi dinamis *health* dan *speed* dari NPC. Sehingga dapat dilihat evolusi dinamis NPC telah sesuai dengan perilaku dan karakteristik dari *player* seperti yang terlihat pada gambar 3.9



Gambar 3.9. Diagram Evolusi Dinamis NPC Dalam Permainan

Dari gambar 3.9 terlihat proses evolusi dinamis NPC yang terjadi selama permainan berlangsung. Terdapat proses inisialisasi karakteristik *player* beserta evaluasi dari range *player* untuk mendapatkan data karakteristik beserta range dari *player* yang digunakan untuk menentukan evolusi dinamis *health* dan *speed* dari NPC, sehingga permainan menjadi menyenangkan karena NPC dapat menyesuaikan kemampuannya dengan kemampuan *player*, baik *player* dengan karakteristik *player* pemula maupun *player* dengan karakteristik *player* handal.

3.3.4 Spawn Awal (Populasi Awal)

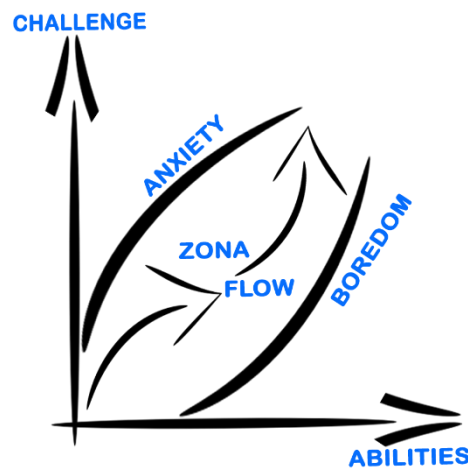
Populasi awal merupakan fase pertama dari model *game* dimana sekumpulan NPC awal dibangkitkan dengan jenis senjata yang acak. Tujuan dari fase ini adalah untuk mendapatkan nilai parameter dari *player* yang digunakan untuk menentukan sekumpulan NPC yang dibangkitkan pada level berikutnya.

Solusi (kromosom) pada NSGA-II direpresentasikan sebagai sekumpulan parameter yang berjumlah 4. Setiap parameter merupakan bilangan real antara 0 hingga 1 yang merepresentasikan tiap-tiap parameter yaitu: *Health (Health Player)*, *Scor*, *NPC missed*, dan *Damage*. Parameter-parameter ini konversi ke dalam nilai tertentu untuk keperluan simulasi evolusi dinamis NPC.

Untuk tujuan penelitian, jumlah populasi diberikan ketetapan yaitu sebanyak 30 solusi Evolusi dinamis pada NPC (kromosom). Namun untuk keperluan simulasi, digunakan jumlah populasi yang bervariasi. Setiap solusi pada populasi awal dibangkitkan secara acak dengan nilai parameter 0 hingga 1.

3.3.5 Dynamic Difficulty Adjustment (DDA)

Tujuan dari pendekatan DDA adalah untuk membuat *game* yang dapat mengimbangi *player* pada setiap level *game*. *Game* dapat dikatakan seimbang jika tantangan dalam *game* sesuai dengan kemampuan *player* seperti model *flow* [21]. Asumsi yang dibangun mengenai *game* yang seimbang dari skenario adalah ketika NPC dapat mengimbangi kemampuan dari *player*. Seperti yang ditunjukkan gambar 3.10



Gambar 3.10. *Flow Channel* Dalam Skenario *Game*

Dalam sebuah *game* dimana NPC memiliki penyerangan yang melampaui kemampuan *player* dapat menyebabkan *game* terasa sulit dimainkan, sehingga dapat menimbulkan keresahan dari *player*, dan begitu pula sebaliknya jika kemampuan *player* melampaui kemampuan NPC dapat membuat *game* menjadi mudah dan memberikan perasaan bosan. Definisi ini merupakan hal yang spesifik dari skenario yang dibangun, namun asumsi yang dibangun ini sudah mencukupi untuk tujuan penelitian.

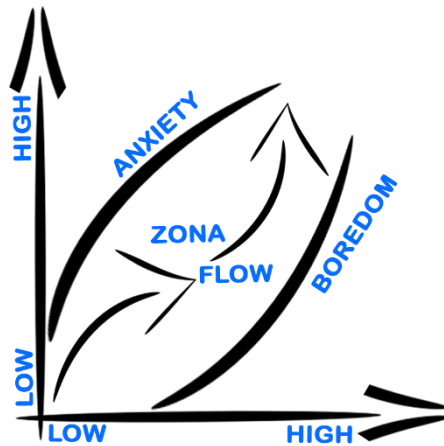
Berdasarkan Gambar 3.10, dipertimbangkan bahwa “*state of flow*” dapat dicapai jika kemampuan dari NPC dapat mengimbangi kemampuan dari *player* dan begitu juga sebaliknya. Berdasarkan kondisi tersebut, DDA terfokus pada pengendalian tipe serangan NPC dengan melihat kondisi akhir dari *player* pada satu level yaitu *health* (*hit point player* (*health player*)), *scor player*, *NPC missed* dan *damage player*. Berdasarkan aturan tersebut dapat dibangun asumsi bahwa “Semakin banyak sisa *health player*, semakin tinggi *scor player*, semakin sedikit *NPC missed* dan nilai *damage player* tinggi maka semakin cepat peningkatan kemampuan NPC pada level berikutnya. Sebaliknya, Semakin sedikit sisa *health player*, semakin rendah *scor player*, semakin banyak *NPC missed* dan nilai *damage player* rendah maka semakin lambat peningkatan kemampuan NPC pada level berikutnya”. Aturan ini dibentuk ke dalam persamaan dengan menambahkan suatu konstanta yang menjadi nilai pengendali parameter NPC di setiap level.

3.3.6 NSGA II

Pengembangan metode optimisasi dalam *game* adalah langkah penting dalam penelitian agar dapat meningkatkan perilaku Evolusi dinamis pada NPC sehingga dapat menyamai dengan perilaku *player*. Nilai fungsi obyektif dari setiap solusi pada populasi dievaluasi. Operator genetika diaplikasikan pada populasi melalui seleksi, persilangan, dan mutasi. Jika kondisi berhenti ditemukan, maka proses optimisasi selesai, atau siklus NSGA-II diulang.

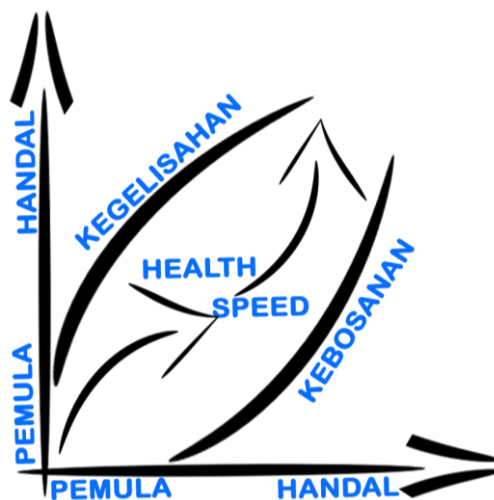
NSGA-II yang merupakan metode optimisasi multiobyektif untuk perilaku Evolusi dinamis pada NPC. Obyektif yang digunakan pada NSGA-II berjumlah 2, namun berada dalam satu domain yaitu evolusi dinamis NPC. Gambar 3.12

menunjukkan flow channel dengan detail NSGA-II pada perilaku evolusi dinamis pada NPC.



Gambar 3.11 Multiobyektif di dalam *Flow Channel*

Bagian berikutnya menjelaskan setiap komponen yang ingin dicapai dalam penelitian ini berdasarkan permasalahan yang dihadapi. Komponen tersebut merupakan perilaku yang menjelaskan karakteristik pemain pemula maupun handal dengan menggunakan pendekatan pada konsep *flow chanel*. Dengan semakin tinggi tingkat kesulitan dari NPC yang dihadapi dapat menimbulkan kegelisahan terhadap *player*, namun jika tingkat kesulitan dari NPC yang dihadapi terlalu mudah. Maka dapat menimbulkan kebosanan terhadap *player*. Karena itu NPC harus dapat menyesuaikan diri terhadap perilaku *player* yang sedang dihadapi.



Gambar 3.12 Konsep Perilaku didalam *Flow Channel*

3.3.6.1 Evaluasi Fungsi Objektif

Pada penelitian ini *evaluate objective function* digunakan untuk dapat melihat perubahan parameter yang terjadi pada *player* disetiap akhir level, perlu didapatkan nilai parameter dari *player* setelah permainan berakhir sebagai parameter *evaluate objective function* serta perhitungan pada NSGA-II, untuk dapat mengoptimisasi evolusi dinamis NPC. Evaluasi obyektif dibutuhkan untuk menghitung nilai obyektif yang nantinya dapat digunakan pada NSGA-II. Evaluasi obyektif dilakukan berdasarkan simulasi (*simulation-based*), karena tidak mungkin untuk mengukur nilai obyektif dari *player* berdasarkan parameter-parameter yang tersedia selama permainan tanpa memperhitungkan kondisi *player* saat bermain (*gameplay*).

Terdapat 4 parameter *evaluate objective function* yang nilainya diambil dari parameter *player* selama permainan berlangsung. Setiap kondisi *player* dapat mempengaruhi nilai *evaluate objective function* yang dipakai pada simulasi (*simulation-based*). Berikut 4 parameter yang berpengaruh terhadap penentuan evolusi dinamis NPC yang optimal terhadap *speed* dan *health* dari NPC:

1. *Score player* : *score* yang didapatkan *player* dalam satu permainan. Nilai parameter dari *score* diambil ketika satu level telah selesai.
2. *Health player* : sisa *health* dari *player* dalam satu permainan. Nilai parameter dari *health* diambil ketika satu level telah selesai.
3. *NPC missed* : *NPC Missed* adalah hasil ketidakmampuan *player* dalam menembak sasaran hingga pesawat NPC dapat melewati *player* tanpa dapat dihancurkan oleh *player*. Nilai parameter dari *NPC Missed* diambil setelah satu level berakhir.
4. *Damage player* : *damage* adalah total tembakan dari *player* yang berusaha menghancurkan pesawat NPC. Nilai parameter dari *damage* diambil setelah satu level telah dilewati oleh *player*.

Dari keempat parameter tersebut maka diperoleh rumus *evaluate objective function* untuk menentukan evolusi dinamis NPC, berikut rumus – rumusnya: Inisialisasi untuk setiap parameter pada *evaluate objective function*:

1. $Health\ player = hp$
2. $Scor\ player = sp$

3. $NPC\ missed = NM$
4. $Damage\ player = dp$

Untuk membangun fungsi objective ini digunakan pendekatan rumus operasi aritmatika biasa, seperti perkalian, penambahan, pengurangan, dan pembagian. Pada pendekatan ini fungsi matematis untuk mendeskripsikan pareto-optimal front diasumsikan ke dalam ruang obyektif. Proses simulasi dilakukan dengan melihat parameter *objective function* dari *player* ketika menghadapi NPC pada level sebelumnya.

Tujuan dari simulasi ini adalah untuk mengetahui berapa *score* yang didapatkan *player*, jumlah *health* yang masih dimiliki *player*, total NPC yang tidak berhasil dihancurkan oleh *player* serta jumlah *damage player* yang digunakan untuk dapat menghancurkan NPC dalam satu kali permainan.

Berdasarkan kondisi simulasi di atas dapat diambil nilai parameter persamaan *objective function* dari *player* sehingga dapat dibuat persamaan untuk *objective function score* sebagai berikut:

$$sp = sd * 10 \quad (3.1)$$

Dengan *sp* adalah variable dari *score player*. Dan *sd* adalah jumlah dari NPC yang dapat dihancurkan oleh *player*. Sehingga *Score player* dapat bertambah 10 apabila *player* dapat menghancurkan pesawat musuh (NPC) dalam satu kali permainan atau dalam satu level.

Objective function selanjutnya adalah *health* dari *player*. Dimana *health player* dapat berkurang apabila pesawat *player* tertembak oleh pesawat NPC. Sehingga dapat dibuat persamaan sebagai berikut:

$$hp = ha - \left(\left(\frac{0,03*a*b}{1+(0,3*(a*b))} \right) * 100\% \right) \quad (3.2)$$

Dengan *hp* adalah variable dari *health player*. Sedangkan *ha* adalah set awal dari *health player*, *a* adalah jumlah tembakan senjata dari NPC yang mengenai pesawat *player* sedangkan *b* adalah damage senjata NPC yang dapat mengurangi sebagian *health player*. Perlu diberikan nilai sebagai nilai kemungkinan yang terjadi pada *healt player* selama permainan berlangsung dalam satu level. Untuk dapat memperoleh nilai keseimbangan antara simulasi dan uji coba nya.

Objective function selanjutnya adalah *NPCMissed* dari *player* dalam satu

kali permainan. Dimana NPC yang tidak berhasil dihancurkan oleh *player* berbanding terbalik dengan NPC yang dapat dihancurkan oleh *player*. Maka dapat diambil persamaan sebagai berikut:

$$NM = (ss - sd)/ss \quad (3.3)$$

Dengan *NM* adalah variable dari NPC *Missed*, *sd* adalah total NPC yang berhasil dihancurkan oleh *player* dalam satu level dan *ss* adalah total NPC yang muncul untuk menghadapi *player* (*spawn* NPC) dalam satu level.

Objective function selanjutnya adalah *damage* dari *player* untuk dapat menghancurkan pesawat NPC sehingga dapat dibuat persamaan sebagai berikut:

$$dp = (c * d) - \left(\left(\frac{0,3*a*b}{1+(0,03(a*b))} \right) * 100\% \right) \quad (3.4)$$

Dengan *dp* adalah variable dari *damage player* untuk dapat menghancurkan pesawat NPC. Sedangkan *a* adalah jumlah tembakan dari senjata NPC untuk menghancurkan pesawat *player*, *b* adalah *damage* dari senjata NPC yang mengenai pesawat *player*, *c* adalah jumlah rata-rata tembakan dari senjata *player* yang mengenai pesawat NPC, *d* adalah *damage* dari senjata *player* yang mengenai pesawat NPC.

Dalam simulasi diperlukan juga persamaan untuk mendapatkan nilai total dari seluruh NPC yang dibangkitkan (*spawn*). Persamaan tersebut adalah *totalNPC* yang dibuat untuk mendefinisikan nilai parameter dari *gen* pertama. Persamaan tersebut adalah sebagai berikut:

$$n = 30 + 1 \quad (3.5)$$

Dengan *n* adalah jumlah dari NPC yang dibangkitkan dalam satu level. Simulasi dalam penelitian ini dilakukan lebih dari satu kali percobaan, dikarenakan bahwa hasil simulasi dalam satu kali percobaan belum dapat menentukan hasil yang sesuai ketentuan (*nondeterministic*).

Sehingga untuk mendapatkan hasil yang sesuai harapan yang dapat digunakan untuk perhitungan NSGA-II adalah berdasarkan nilai rata-rata dari sejumlah simulasi pada tiap individu dalam populasi.

3.3.6.2 Fungsi Objective

Pendekatan yang digunakan untuk membangun fungsi obyektif adalah *Bottom-Up Approach* [22]. Pada pendekatan ini fungsi matematis untuk

mendeskripsikan *pareto-optimal front* diasumsikan ke dalam ruang obyektif dan keseluruhan ruang pencarian obyektif dibangun berdasarkan front ini. Pada penelitian ini, kondisi *Pareto-optimal front* adalah dimana semua fungsi obyektif adalah perbandingan dari tiap parameter evolusi dinamis NPC (*speed* (f_1) dan *health* (f_2)). Dari tiap individu dicari nilai *Pareto-optimal front* dengan hasil simulasi tiap individu NPC terhadap parameter *objective function* dari perilaku *player*.

Pada penelitian ini fungsi yang dioptimalkan adalah perilaku adaptif dari NPC untuk menghadapi perilaku *player*, dimana tujuan yang ingin dicapai adalah dengan memaksimalkan dan meminimumkan Evolusi dinamis NPC (*speed* (f_1) dan *health* (f_2)).

Agar *player* tidak mudah bosan karena NPC mudah dihadapi atau terlalu sulit untuk dihadapi. Maka di tentukan *objective function* untuk dapat mengatur Evolusi dinamis NPC tersebut.

Peneliti mencoba membuat sekenario *game* untuk melihat perilaku evolusi dinamis NPC. Dimana dapat diasumsikan sebagai evolusi dinamis *speed* dan *health* dari NPC untuk mengimbangi perilaku *player*.

Gen pertama yang dijadikan *objective function* untuk evolusi dinamis NPC adalah *gen speed*, dimana *gen speed* adalah parameter untuk meningkatkan pergerakan dari NPC. *Speed* adalah salah satu *objective function* yang menjadi tujuan pertama dalam simulasi menggunakan NSGA-II. *Speed* disimbolkan dengan f_1 . Nilai fungsi *objective* dari parameter *player* seperti *score player*, *health player*, *NPCmissed*, dan *damage player* digunakan untuk mengontrol perubahan parameter *speed* yang tepat dalam *game*. Untuk mengontrol evolusi dari *objective function* *speed* maka peneliti membuat nilai parameter untuk merubah evolusi kecepatan dari NPC yang kemudian dapat disebut sebagai *speed evolution* (*se*). Maka dapat dibuat persamaan untuk menentukan *objective function* dari *speed player* sebagai berikut:

$$f_1 = \left(\frac{(150 * genScore + 0.5 * genDamage)}{2} \right) * \frac{gs}{w} * \left(\frac{genScore * genDamage}{2} \right) * n * se + \left(\frac{hp * dp}{2} \right) \quad (3.6)$$

Dengan f_1 adalah *objective function* dari *speed* NPC, sedangkan *gs* adalah *Speed* awal dari NPC, *n* adalah jumlah NPC yang dibangkitkan, *hp* adalah nilai parameter dari *health player*, *dp* adalah nilai parameter dari *damage player*, dan *se* adalah variable dari parameter *speed evolution* yang digunakan untuk mengontrol

evolusi dari *speed* NPC. Terdapat juga *genScore* dengan rentang nilai *gen* yang dibangkitkan pada individu antara 0 sampai dengan 1, dan *genDamage* dengan rentang nilai *gen* yang dibangkitkan pada individu antara 0 sampai dengan 1. Terdapat juga faktor penimbang dari *gen* yang dibangkitkan (*weight*) dari *gen* kekuatan *player* yang bertujuan untuk menormalisasi parameter kekuatan dari *player* sehingga total dari *gen* yang menjadi parameter kekuatan dari perilaku *player* adalah 1 dengan cara menjumlahkan keempat *gen* parameter kekuatan. Sehingga dapat dibuat persamaan penimbang kekuatan *player* sebagai berikut:

$$w = \text{genScore} + \text{genHealth} + \text{genNPCmissed} + \text{genDamage} \quad (3.7)$$

Dengan w adalah *weight* sebagai penimbang kekuatan *gen* yang dibangkitkan dari parameter *player*, keempat *gen* tersebut adalah individu yang dibangkitkan dengan rentang nilai *gen* antara 0 sampai dengan 1 untuk keperluan simulasi.

Gen kedua yang dijadikan *objective function* untuk evolusi dinamis NPC adalah *gen health*, dimana *gen health* adalah parameter untuk meningkatkan nyawa dari NPC. *Objective function* dari *gen* kedua (*health*) adalah untuk menambahkan jumlah *health* untuk NPC. *Health* adalah salah satu *objective function* yang menjadi tujuan kedua dalam simulasi menggunakan NSGA-II. *Objective function health* disimbolkan dengan f_2 . Nilai *objective function* dari parameter *player* seperti *score player*, *health player*, *NPCmiss*, dan juga *damage player*. Keempat *objectif function* digunakan untuk menentukan evolusi dinamis dari perubahan parameter pada *objective function health* pada NPC. Sedangkan untuk mengontrol perubahan parameter pada *objective function health* yang tepat dalam *game* kemudian dapat disebut sebagai *health evolution* (*he*). Sehingga dapat dibuat rumus persamaan untuk menentukan evolusi dinamis *health player* sebagai berikut:

$$f_2 = \left(\frac{(150 * \text{genScore} + 0.7 * \text{genNPCmiss} + 0.5 * \text{genDamage})}{3} \right) * \frac{\text{genHealth}}{w} * \left(\left(\frac{\text{genScore}}{\text{genNPCmiss}} - \text{genDamage} \right) / 3 \right) * n * he + \left(\frac{hp * NM * dp}{2} \right) \quad (3.8)$$

Dengan f_2 adalah symbol dari parameter evolusi dinamis dari *health* NPC, *he* adalah symbol dari fungsi yang digunakan untuk dapat mengontrol evolusi dinamis dari *health* NPC (*health evolution*), n adalah jumlah NPC yang

dibangkitkan, *hp* adalah *health player*, *NM* adalah *NPCMissed*, *dp* adalah *damage player* yang digunakan untuk menghancurkan NPC. terdapat juga *genScore*, *genHealth*, *genNPCMissed*, dan *genDamage*, keempat *gen* tersebut adalah individu yang dibangkitkan dengan rentang nilai *gen* antara 0 sampai dengan 1 untuk keperluan simulasi.

Terdapat juga faktor penimbang (*weight*) dari kekuatan *player* yang bertujuan untuk menormalisasi parameter kekuatan dari *player* sehingga total dari *gen* yang menjadi parameter kekuatan dari perilaku *player* adalah 1 dengan cara menjumlahkan ketiga *gen* parameter kekuatan. Sehingga dapat dibuat persamaan penimbang kekuatan *player* sebagai berikut:

$$w = \text{genScore} + \text{genHealth} + \text{genNPCmissed} + \text{genDamage} \quad (3.9)$$

Dengan *w* adalah *weight* sebagai penimbang kekuatan *gen* yang dibangkitkan dari parameter *player*, terdapat juga *genScore*, *genHealth*, *genNPCMissed*, dan *genDamage*, dimana keempat *gen* tersebut adalah individu yang dibangkitkan dengan rentang nilai *gen* antara 0 sampai dengan 1 untuk keperluan simulasi.

3.3.6.3 Pengurutan Nondominasi (*Nondominated Sort*)

Suatu individu dapat dikatakan mendominasi individu lainnya jika memenuhi aturan-aturan berikut:

1. Individu A tidak lebih buruk ketimbang individu B pada semua obyektif ($A \geq B$).
2. Individu A setidaknya memiliki satu obyektif lebih baik ketimbang individu B.

Berdasarkan aturan tersebut, setiap individu dibandingkan dengan individu lainnya di dalam satu populasi. Permasalahan optimisasi dalam kasus ini adalah mencari nilai maksimum dari 4 obyektif sehingga kondisi dominasi adalah jika suatu individu tidak lebih buruk dari individu lainnya dan setidaknya memiliki satu obyektif yang nilainya lebih besar daripada individu lainnya (Obyektif A.1 == Obyektif B.1,...,Z.1 AND Obyektif A.2 == Obyektif B.2,...,Z.2 AND Obyektif A.3 > Obyektif B.3,...,Z.3). Individu tersebut dapat berada pada front pertama. Kemudian untuk mengisi front berikutnya berdasarkan individu yang didominasi oleh front sebelumnya.

3.3.6.4 Crowding Distance

Setelah pengurutan berdasarkan nondominasi dilakukan, maka perhitungan crowding distance dilakukan. Individu dipilih berdasarkan peringkat dan crowding distance, maka semua individu pada populasi diberikan nilai crowding distance. Crowding distance tiap individu dihitung berdasarkan front individu tersebut berada, sehingga membandingkan crowding distance antara dua individu yang berbeda front adalah hal yang tidak berguna.

Ide dasar dibalik crowding distance adalah menemukan euclidian distance antara setiap individu pada front berdasarkan jumlah obyektifnya pada ruang dimensi sesuai dengan jumlah obyektif. Sebagai contoh, dalam permasalahan ini terdapat 3 obyektif, maka ruang dimensinya adalah 3 pula. Persamaan untuk mendapatkan crowding distance adalah sebagai berikut:

$$I(d_k) = I(d_k) + \frac{I(k+1).m - I(k-1).m}{f_m^{max} - f_m^{min}} \quad (3.10)$$

Dengan $I(d_k)$ adalah crowding distance dari individu ke-k, k adalah indeks individu pada front F_i yang dimulai dari 2 hingga n-1. Untuk $I(d_1)$ dan $I(d_n)$ yang menjadi nilai batas untuk individu pada front F_i diberikan nilai tak hingga (∞). $I(k).m$ merupakan nilai dari fungsi obyektif ke-m pada individu ke-k, sedangkan f_m^{max} dan f_m^{min} secara berturut adalah fungsi obyektif terbesar dan terkecil dari fungsi obyektif ke-m pada front yang dilakukan crowding distance. Untuk $I(d_1)$ dan $I(d_n)$ yang menjadi nilai batas untuk individu pada front F_i diberikan nilai tak hingga (∞).

3.3.6.5 Seleksi

Saat individu telah diurutkan berdasarkan nondominasi dan crowding distance telah dihitung. Seleksi dilakukan dengan *crowded-comparison-operator* (\prec_n). Perbandingan (*comparison*) dilakukan dengan aturan sebagai berikut:

1. Peringkat nondominasi (p_{rank}). Sebagai contoh, jika individu pada front F_i akan memiliki peringkat $p_{rank} = i$.
2. Crowding Distance $F_i(d_j)$

$$p \prec_n q \text{ jika:}$$
 - $p_{rank} < q_{rank}$

- atau jika p dan q berada pada front yang sama (F_i) maka dicari perbandingan crowding distance $F_i(dp) > F_i(dq)$.

3.3.6.6 Operator *Genetika*

Pada penelitian ini, nilai yang digunakan pada NSGA-II bertipe *Real* sehingga persilangan yang digunakan adalah Simulated Binary Crossover (SBX) [23] dan mutasi yang digunakan polynomial mutation [24].

1. *Simulated Binary Crossover.*

Setiap individu baru pada populasi berikut berasal dari sepasang individu dari generasi saat ini yang mengalami persilangan (*crossover*). Proses ini akan menghasilkan dua *offspring* atau individu “anak”. Ide dasar dari proses persilangan ini adalah *offspring* akan membawa karakteristik dari dua individu “orang tua” ke generasi berikutnya. Metode persilangan yang digunakan pada penelitian ini adalah *Simulated Binary Crossover*.

Proses yang dilakukan SBX adalah menyilangkan bilangan real dengan mensimulasikan seperti persilangan biner yang diobservasi secara alami dan ditunjukkan sebagai berikut.

$$\begin{aligned} o_{1,k} &= \frac{1}{2}[(1 - \beta_k)p_{1,k} + (1 + \beta_k)p_{2,k}] \\ o_{2,k} &= \frac{1}{2}[(1 + \beta_k)p_{1,k} + (1 - \beta_k)p_{2,k}] \end{aligned} \quad (3.11)$$

Dengan $o_{i,k}$ adalah *offspring* i^{th} dengan komponen k^{th} . $p_{i,k}$ adalah *parent* terpilih dan $\beta_k (\geq 0)$ adalah sampel dari angka yang dibangkitkan secara acak dengan kepadatan

$$\begin{aligned} p(\beta) &= \frac{1}{2}(\eta_c + 1)\beta^{\eta_c}, \text{ if } 0 \leq \beta \leq 1 \\ p(\beta) &= \frac{1}{2}(\eta_c + 1)\frac{1}{\beta^{\eta_c+2}}, \text{ if } \beta > 1 \end{aligned} \quad (3.12)$$

Distribusi ini bisa didapatkan dari angka acak yang disampel secara *uniform* u berkisar antara (0,1). η_c adalah indeks distribusi dari persilangan. Indeks distribusi ini bertujuan untuk mengukur seberapa baik penyebaran dari *offspring* dari *parent*-nya. η_c yang digunakan pada penelitian ini sebesar 20. Probabilitas persilangan yang digunakan adalah 1.

2. *Polynomial Mutation*

Mutasi adalah teknik yang umum digunakan pada algoritma *genetika* untuk menjaga keragaman *genetik* di dalam populasi saat melalui generasi.

Tingkat mutasi yang terlalu rendah akan mengarahkan pada kondisi di mana setiap individu akan mirip dan keragaman solusi akan hilang. Terlalu besarnya tingkat mutasi maka dapat menyebabkan hilangnya solusi yang baik.

Penelitian ini menggunakan suatu model mutasi yang menerapkan model probabilitas kepada solusi. Model mutasi ini disebut dengan *Polynomial Mutation*.

$$o_k = p_k + (p_k^u - p_k^l)\delta_k \quad (3.13)$$

Dengan o_k adalah offspring dan p_k adalah parent dengan p_k^u menjadi batas atas pada komponen *parent*. p_k^l adalah batas bawah dan δ_k adalah variasi kecil yang mana dikalkulasi dari distribusi polinomial menggunakan

$$\begin{aligned} \delta_k &= (2r_k)^{\frac{1}{\eta_m+1}} - 1, \text{ if } r_k < 0.5 \\ \delta_k &= 1 - [2(1 - r_k)]^{\frac{1}{\eta_m+1}}, \text{ if } r_k \geq 0.5 \end{aligned} \quad (3.14)$$

r_k adalah angka acak yang disampel secara uniform berkisar antara (0,1) dan η_m adalah indeks distribusi mutasi. η_m yang digunakan pada penelitian ini adalah 20 dan probabilitas mutasi adalah $1/n$ dimana n adalah ukuran populasi.

3.4 Skenario Simulasi Dalam Permainan

Selanjutnya adalah menentukan skenario simulasi evolusi dimanis NPC dalam *game space shooter* seperti yang terlihat pada gambar 8. Dimana pada level pertama terdapat *non-player character* (NPC) yang muncul (*spawn*) secara bergantian hingga berjumlah total 30 *non-player character* (NPC), setelah seluruh NPC keluar maka sebelum dapat naik kelevel selanjutnya, terlebih dahulu *player* harus menghancurkan BOS penjaga untuk setiap levelnya. *Spawn non-player character* pada level pertama adalah dengan senjata *random* dimana pada level pertama ini tidak menutup kemungkinan *spawn* NPC menggunakan ke empat senjata (*gun, missile, rudal* atau *suicide*).

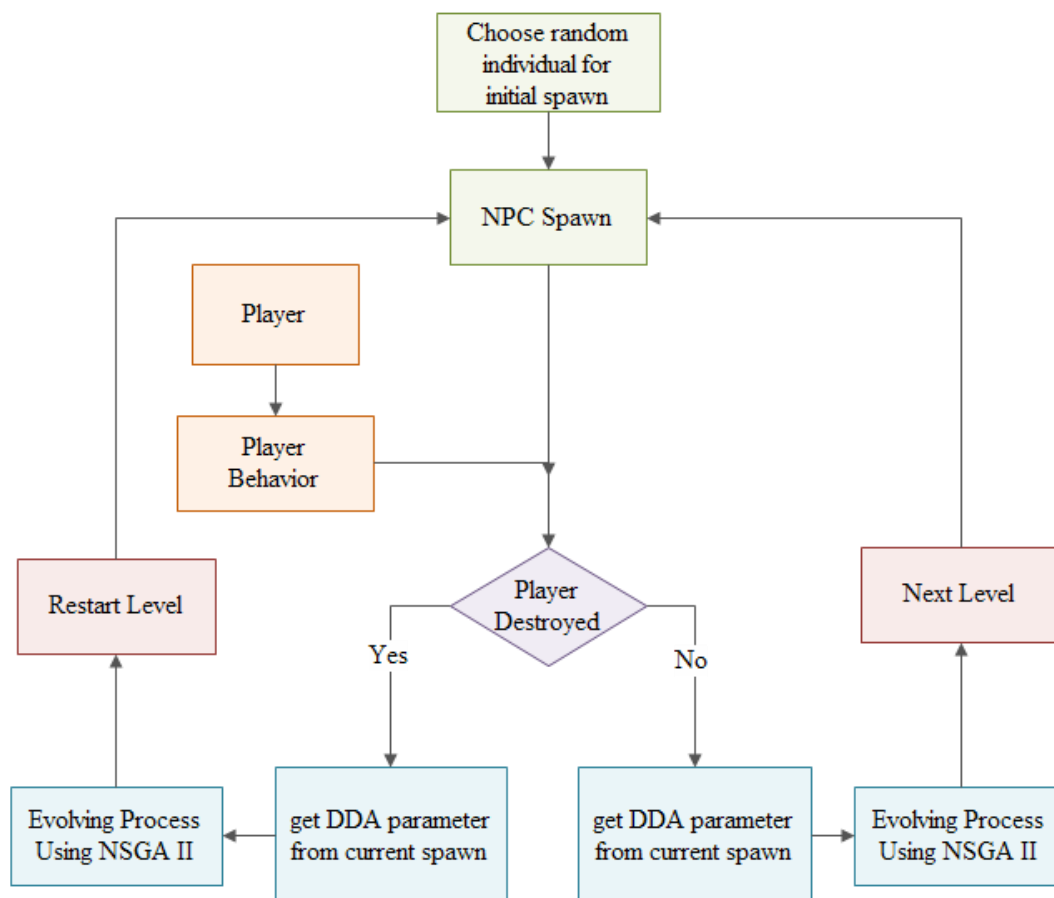
Dari level pertama tersebut maka dapat diambil data untuk menentukan populasi *non-player character* pada level berikutnya, dimana tidak menutup kemungkinan tingkat kesulitan yang dihadapi *player* dapat meningkat secara dratis atau menurun secara dratis, bahkan dapat seimbang dengan kemampuan *player*

tergantung parameter dari *player* pada level sebelumnya.

NSGA-II digunakan untuk menentukan optimasi-optimasi pada parameter *objective function* dari NPC. Dalam penelitian ini *objective function* yang dimaksud adalah *speed* dan *health* dari NPC. Dimana dapat dianalogikan seperti berikut:

Selama permainan berlangsung, parameter dari NPC seperti *speed* (f_1) dan *health* (f_2) dapat berubah sesuai dari parameter *player*. Karena kedua gen ini digunakan untuk evolusi dinamis NPC untuk menghadapi *player*.

Pada state awal *game*, *spawn* awal NPC dapat dianalogikan sebagai *default*, sebagai ujicoba tingkat kemampuan *player* pada level pertama dalam permainan. Setelah level pertama selesai, maka parameter yang ada pada *player* (*score player*, *health player*, *NPCMissed*, dan *damage player*) dihitung untuk menentukan evolusi dinamis NPC, dimana NPC yang dibangkitkan pada level selanjutnya menggunakan senjata *gun*, *missile*, *rudal*, atau *suicide* beserta evolusi parameter pada NPC (*speed* (f_1) dan *health* (f_2)).



Gambar 3.13. Blog Diagram *Game* dengan DDA dan NSGA II

Pada gambar 3.8 adalah blog diagram dari simulasi menggunakan NSGA II, dimana NSGA II digunakan untuk simulasi yang dapat menghasilkan beberapa solusi *objective*. Dalam simulasi pada permainan, *player* akan bermain selama satu level, dan pada level berikutnya NSGA II akan dipakai untuk menghasilkan solusi optimal untuk level berikutnya dengan melihat *objective function* dari *player*. *Objective function* dari parameter *player* digunakan untuk menemukan solusi optimal untuk evolusi dinamis *speed* dan *health* dari NPC.

Individu-individu dibangkitkan dengan *gen random* pada level pertama, dimana parameter dari NPC cenderung masih dalam keadaan konstan, tanpa terjadi evolusi dinamis *speed* dan *health* dari NPC. Level pertama hanya sebagai data *training* untuk dapat melihat perilaku atau *objective function* dari *player* untuk dipakai sebagai data *training* untuk melakukan evolusi dinamis *speed* dan *health* dari *player*.

3.5 Skenario Pengujian

Pada skenario pengujian peneliti menggunakan dua pengujian, pengujian pertama menggunakan uji coba pada tool unity untuk melihat dan pengambil data *training* dari permainan *player* ketika menghadapi NPC disetiap levelnya. Untuk mencari data *training* sebagai data untuk evolusi dinamis NPC yang sedang dihadapi *player* pada setiap levelnya. Pada pengujian kedua menggunakan tool matlab untuk melakukan uji coba dengan metode NSGA II untuk melihat grafik dan pola penyebaran kromosom dari evolusi dinamis NPC.

3.5.1 Skenario Pengujian Pada Permainan

Scenario pengujian dilakukan guna mengetahui apakah proses dari metode yang diterapkan telah sesuai dengan hasil yang diharapkan. Dalam skenario pengujian ini, dilakukan pengujian evolusi dinamis *non-player character* dengan melihat parameter *objective function* yang terdapat pada *player*, seperti *score player*, *health player*, *NPCMissed*, dan *damage* dari *player* yang digunakan untuk menghancurkan pesawat NPC.

Pengujian dilakukan dengan mengumpulkan beberapa pemain untuk memainkan *game* secara bergantian. Hal tersebut dilakukan untuk mendapatkan

data *training* dari beberapa *player* untuk melihat seberapa jauh *player* dapat menyelesaikan *game*. Oleh karena itu data *training* yang diambil dibedakan menjadi dua. Data *training* dari pemain handal dan data *training* dari pemain lemah.

Setelah data *training* dari beberapa *player* didapatkan, data *training* tersebut diolah untuk mengetahui level terjauh dari masing-masing *player*. Disamping itu pentingnya data *training* tersebut adalah untuk mendapatkan data *training* dari *score player*, *health player*, *NPCMissed* dan *damage* dari *player* yang digunakan untuk menghancurkan NPC dalam satu level permainan untuk mengetahui *objective function* dari masing-masing *player*. Data *training objective function* dari *player* digunakan untuk uji simulasi menggunakan NSGA II, sehingga dapat diperoleh solusi *multi-objective* untuk evolusi dinamis *speed* dan *health* dari NPC.

3.5.2 Skenario Simulasi menggunakan NSGA II

Sebuah skenario pengujian penggunaan NSGA II dilakukan guna mengetahui apakah proses dari metode yang diterapkan telah sesuai dengan hasil yang diharapkan. Dalam skenario pengujian ini, simulasi evolusi dinamis pada NPC dilakukan dengan melihat parameter-parameter *objective function* yang terdapat pada *player*, seperti *score player*, *health player*, *NPCmissd* dan *damage player* untuk menghancurkan pesawat NPC. Data *objective function* dari *player* digunakan untuk menentukan evolusi dinamis *speed* dan *health* dari NPC sehingga dapat dilihat tingkat evolusi yang terjadi pada *speed* dan *health* NPC untuk dapat mengimbangi kemampuan *player*.

Pengujian Performa NSGA-II dilakukan untuk melihat perilaku evolusi dinamis pada NPC. Pengujian difokuskan pada evolusi dinamis *speed* dan *health* dari NPC. Apakah terjadi peningkatan atau penurunan evolusi yang signifikan pada *gen speed* atau *gen health*, ataukah terjadi peningkatan atau penuruna evolusi yang signifikan pada kedua *gen* (*speed* dan *health*) secara bersamaan.

Pengujian performa NSGA II dilakukan dengan simulasi pada beberapa populasi dan generasi secara berurutan. Untuk mendapatkan nilai *objective function* yang sesuai sehingga dapat diperoleh solusi *multi-objective* yang terbaik. Pengujian performa tersebut antara lain:

- a. Simulasi untuk melihat performa NSGA-II untuk melihat evolusi dinamis NPC yang difokuskan pada evolusi *speed* NPC

Pengujian Peforma NSGA-II digunakan untuk mendapatkan solusi *muti-objective* yang optimal untuk evolusi dinamis NPC. Sehingga dapat diperoleh nilai evolusi dinamis yang sesuai dengan kriteria yang diharapkan peneliti. Sehingga tidak terjadi ketidakseimbangan permainan antara *player* dan NPC.

Halaman ini sengaja dikosongkan

BAB IV

HASIL DAN PEMBAHASAN

Bab ini memberikan penjelasan mendalam mengenai pengujian yang dilakukan untuk mengevaluasi pendekatan yang dikembangkan dan memberikan hasil yang didapatkan disertai dengan analisa. Dua model pengujian dilakukan pada bab ini, yaitu:

1. Pengujian 1: Simulasi dalam permainan (*game space shooter*)
2. Pengujian 2: Simulasi untuk melihat performa NSGA-II

Pengujian 1 melibatkan simulasi dalam *game* dengan partisipasi *player*. Dimana *player* dibagi dalam dua golongan, yaitu *player* handal dan *player* pemula. Namun dalam hal ini dibuat beberapa skenario *game* untuk menguji evolusi dinamis *speed* dan *health* dari NPC dengan melihat *objective function* dari perilaku *player*. Pengujian 2 melibatkan beberapa kali percobaan untuk mendapatkan konfigurasi NSGA-II yang ideal melalui simulasi dan melihat performa dari NSGA-II selama simulasi dilakukan. Hasil yang dianggap paling baik dan efisien digunakan di dalam *game* pada level berikutnya.

4.1 Simulasi Tanpa DDA Dan Dengan DDA

Pengujian ini menggunakan test-bed *space shooter* yang sudah dibuat untuk keperluan penelitian ini. Alur pengujian pada *game space shooter* ini dengan membuat dua tipe *game* yaitu dengan menggunakan DDA (Dynamic Difficulty Adjusment) dan tidak menggunakan DDA (Dynamic Difficulty Adjusment).

Terdapat 10 level uji coba pada penelitian ini, dimana pada setiap levelnya kromosom NPC dapat selalu berubah-ubah sesuai dengan perilaku *player*. Akan ada peningkatan maupun penurunan kemampuan dari NPC sesuai dengan parameter *objective function* yang dimiliki oleh pesawat *player* yang didapatkan pada level sebelumnya untuk dapat mengevolusi parameter *objective function* dari NPC. Parameter *objective function* tersebut antara lain *speed* (f_1) dan *health* (f_2) dari pesawat NPC.

Namun pada level pertama pada *game space shooter*, ke empat varian senjata dapat keluar dengan *gen speed* dan *gen health* yang masih konstan. Dikarenakan

pada level pertama, hanya sebagai data test terhadap perilaku *player*, sehingga pada level kedua hingga level ke sepuluh, nilai konstan dari *speed* dan *health* dari NPC dapat berubah-ubah sesuai perilaku *player*.

4.1.1 Simulasi Permainan Tanpa DDA (Dinamic Difficulty Adjusment) dan Evolusi Dinamis NPC

Simulasi permainan tanpa menggunakan DDA (Dinamic Difficulty Adjusment) dimulai pada status NPC yang terus-menerus *spawn* secara random pada *game Space Shooter* di unity. Dimana pada test-bed ini NPC dapat terus menerus keluar dengan berbagai kemungkinan senjatanya (*gun*, *missile*, *rudal*, dan *suicide*) hingga *player* kalah atau *player* berhasil mengalahkan semua musuh.



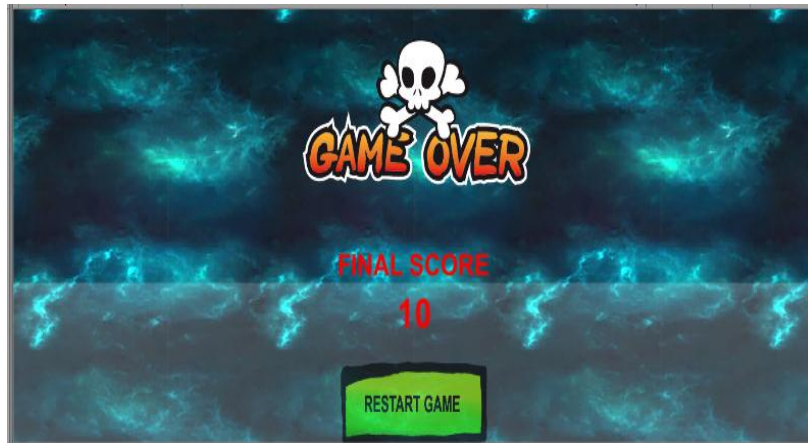
(a)



(b)

Gambar 4.1 Simulasi Permainan Tanpa DDA (a) & (b) kondisi *spawn* player ketika memulai *game*

Pada setiap perulangan level ketika *player* kalah maupun menang, *game* selalu melakukan perulangan dengan musuh yang selalu *spawn* tanpa adanya perubahan yang berarti. Meskipun terdapat leveling dalam *game*, perubahan tingkat kesulitan yang statis dapat membuat *player* mudah bosan karena *game* terlalu sulit dimainkan, ataupun *game* terlalu mudah diselesaikan oleh *player*



Gambar 4.2 Kondisi User Interface Ketika *Player* Kalah



Gambar 4.3 Kondisi User Interface Ketika *Player* Menang

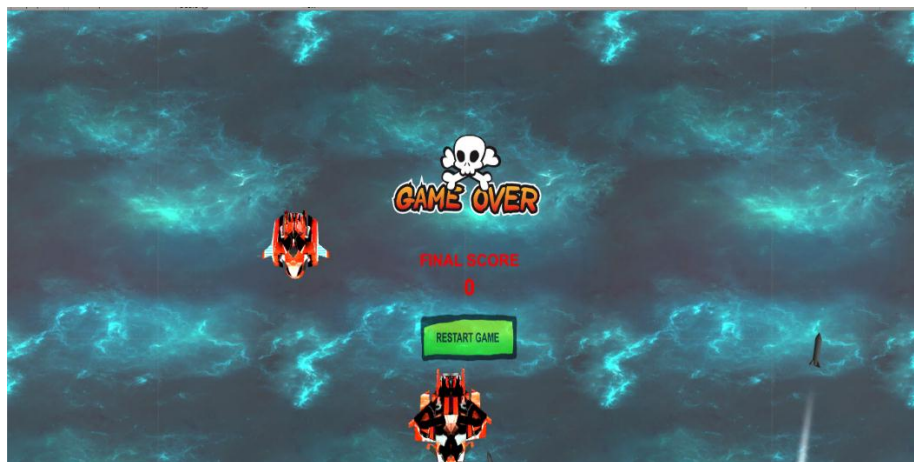
4.1.2 Simulasi Permainan Dengan DDA (Dinamic Difficulty Adjusment) dan Evolusi Dinamis NPC

Simulasi permainan menggunakan DDA (Dinamic Difficulty Adjusment) dimulai pada status perilaku evolusi dinamis pada NPC secara random yang ditandai dengan status TBD pada *game space shooter* di unity.



Gambar 4.4 Simulasi Permainan Pada Status TBD

Setelah *game* berlangsung dalam satu level maka parameter *objective function* dari *player* dihitung untuk menentukan perilaku evolusi dinamis pada NPC untuk level berikutnya. Jika *player* kalah dalam satu *game*, maka pada level berikutnya tingkat kesulitan *game* kembali menjadi *easy*.



(2)

Gambar 4.5 Menunjukkan Perubahan Level Setelah Satu Kali Level

Jika dalam satu level berikutnya *player* dapat mengalahkan NPC yang sudah beregenerasi dengan evolusi dinamis NPC maka parameter *objective function* dari *player* dihitung kembali untuk menentukan tingkat evolusi dinamis pada NPC untuk level berikutnya. Apakah perubahan pada level berikutnya tetap *easy*, *medium* atau *hard*. Dilihat dari *objective function* dari parameter yang terdapat pada *player*



Gambar 4.6 Menunjukkan Perubahan Level Setelah Satu Kali *game*

4.2 Simulasi Pada Permainan

Uji coba pada *game* dibuat dalam beberapa skenario dan aturan untuk menjaga keseimbangan antara *player* dan evolusi dinamis NPC pada *game*. Pengujian ini menggunakan test-bed *game* dengan *genre* FPS. *Game genre* FPS yang dipilih adalah *game space shooter* yang dibuat sendiri oleh peneliti untuk keperluan penelitian ini (Sub-bab 3.2). Alur pengujian pada *game space shooter* ini adalah mencari dua tipe *player* yang dapat mewakili *player* pemula dan *player* handal untuk mencoba *game* yang telah dibuat oleh peneliti, untuk mendapatkan data training dari tipe *player* pemula dan tipe *player* handal untuk digunakan sebagai data training dalam simulasi evolusi dinamis *speed* dan *health* dari NPC.

Berdasarkan hasil uji coba dengan menjalankan *game space shooter* untuk mendapatkan hasil dari *objective function* dari parameter yang dimiliki oleh *player*

(*score player*, *health player*, *NPC missed*, dan *damage* dari *player* untuk menghancurkan pesawat NPC) terhadap evolusi dinamis NPC selama *game* berlangsung. *Objective function* dari parameter *player* dihitung kembali berdasarkan setiap level yang telah diselesaikan oleh *player*. Seperti yang ditunjukkan pada Gambar 4.5 menghasilkan parameter seperti yang ditunjukkan pada Tabel 4.1 dan Tabel 4.2.



Gambar 4.7 Simulasi Uji Permainan

4.2.1 Simulasi Permainan Pada *Player* Pemula

Simulasi permainan pada *player* pemula adalah untuk mengetahui *objective function* data *player* dalam satu permainan ketika menghadapi evolusi dinamis NPC pada setiap levelnya. Sehingga diperoleh data *training* untuk mendapatkan *objective function* dari *player* pemula dalam satu level.

Selama *player* bermain, data *training* diperoleh untuk melihat perubahan parameter *objective function* dari *player* selama permainan berlangsung dalam setiap level yang telah dilewati oleh *player*. Data *training objective function* dari parameter *player* selanjutnya dapat diolah sebagai uji coba simulasi NSGA II. Untuk menemukan solusi *multi-objective* yang optimal dalam menemukan *zona flow* bagi *player* pemula. Data *training* parameter *objective function* dari *player* pemula dapat dilihat pada tabel 4.1

Tabel 4.1 Data Pemain Pemula Dalam Satu *Game*

<i>Game</i>	<i>Level</i>	<i>Level Game</i>	<i>Health Player</i>	<i>Scor Player</i>	<i>NPC Missed</i>	<i>Damage Player</i>
1	1	TDB	50	100	1	93
2	2	<i>Easy1</i>	65	200	10	125
3	3	<i>Easy3</i>	70	300	13	80
4	4	<i>Medium1</i>	0	380	5	310
5	1	<i>Easy1</i>	100	100	0	230
6	2	<i>Hard1</i>	0	150	5	313
7	1	<i>Easy1</i>	-	-	-	-

Dari data simulasi permainan *player* pemula yang ditunjukkan pada tabel 4.1 diperoleh data evolusi dinamis *speed* dan *health* dari NPC. Pada hasil perubahan yang ditunjukkan pada Tabel 4.2 untuk *player* pemula terlihat bahwa pada awal permainan parameter evolusi dinamis NPC mengalami perubahan yang sangat pelan karena batasan dari pemain pemula sehingga perubahan parameter berubah perlahan disebabkan factor perilaku dan karakteristik dari *player*

Tabel 0.2 Hasil Evolusi Dinamis NPC Pada Parameter Pemain Pemula

Game	Level	Speed	Health
1	1	0	0
2	2	11	110
3	3	13	130
4	4	14	140
5	1	11	110
6	2	17	170
7	1	11	110

4.2.2 Simulasi Permainan pada *Player* Handal

Simulasi permainan pada *player* handal digunakan untuk mengetahui data *training* dari *objective function* pada parameter *player* dalam satu permainan ketika

menghadapi evolusi dinamis NPC pada setiap levelnya. Sehingga diperoleh data *training* untuk mendapatkan *objective function* dari *player* yang berguna untuk menemukan *zona flow* bagi *player* handal untuk level berikutnya.

Selama *player* bermain, data *training* diperoleh untuk melihat perubahan parameter *objective function* dari *player* selama permainan berlangsung dalam setiap level yang telah dilewati oleh *player*. Data *training* dari parameter *objective function* selanjutnya dapat diolah sebagai uji coba simulasi menggunakan NSGA II. Untuk menemukan solusi *multi-objective* yang optimal dalam menemukan *zona flow* bagi *player* handal. Data *training* dari parameter *objective function* *player* handal dapat dilihat pada tabel 4.3.

Tabel 4.3 Data Pemain Handal Dalam Satu *Game*

<i>Game</i>	<i>Level</i>	<i>Level Game</i>	<i>Health Player</i>	<i>Scor Player</i>	<i>NPCMissed</i>	<i>Damage Player</i>
1	1	TDB	75	100	2	50
2	2	Medium1	30	200	5	160
3	3	Easy3	85	300	11	230
4	4	Hard1	90	400	7	490
5	5	Hard2	0	430	15	320
6	1	Easy1	100	100	4	30
7	2	Hard1	-	-	-	-

Dari data simulasi permainan *player* handal yang ditunjukkan pada tabel 4.4 diperoleh data evolusi dinamis *speed* dan *health* dari NPC. Pada hasil perubahan yang ditunjukkan pada Tabel 4.4 untuk *player* handal terlihat bahwa pada awal permainan parameter evolusi dinamis NPC mengalami perubahan yang sangat cepat karena karakteristik dari *player* handal sehingga perubahan parameter berubah terlalu cepat disebabkan factor perilaku dan karakteristik dari *player*

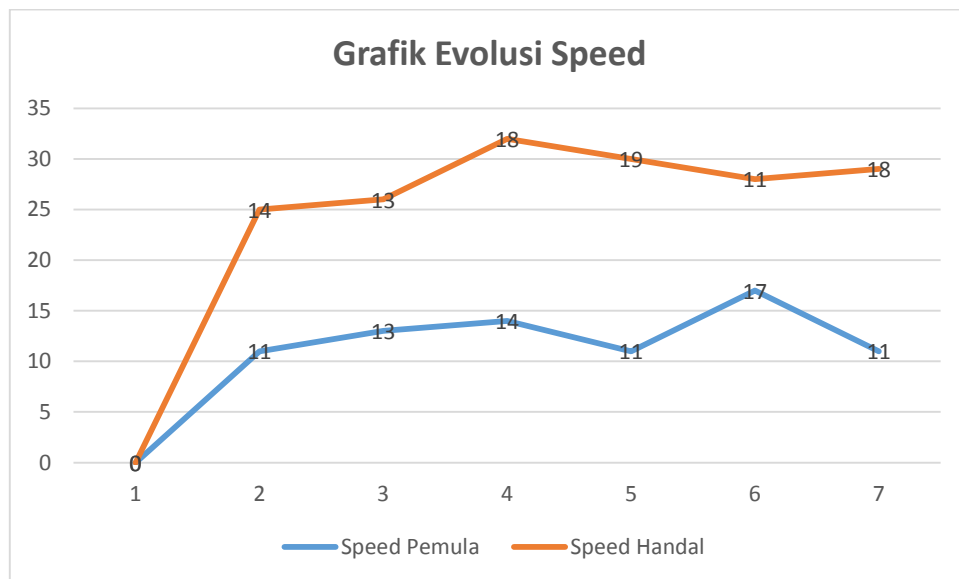
Tabel 0.4 Hasil Evolusi Dinamis NPC Pada Parameter Pemain Handal

<i>Game</i>	<i>Level</i>	<i>Speed</i>	<i>Health</i>
1	1	0	0
2	2	14	140

3	3	13	130
4	4	18	180
5	5	19	190
6	1	11	110
7	2	18	180

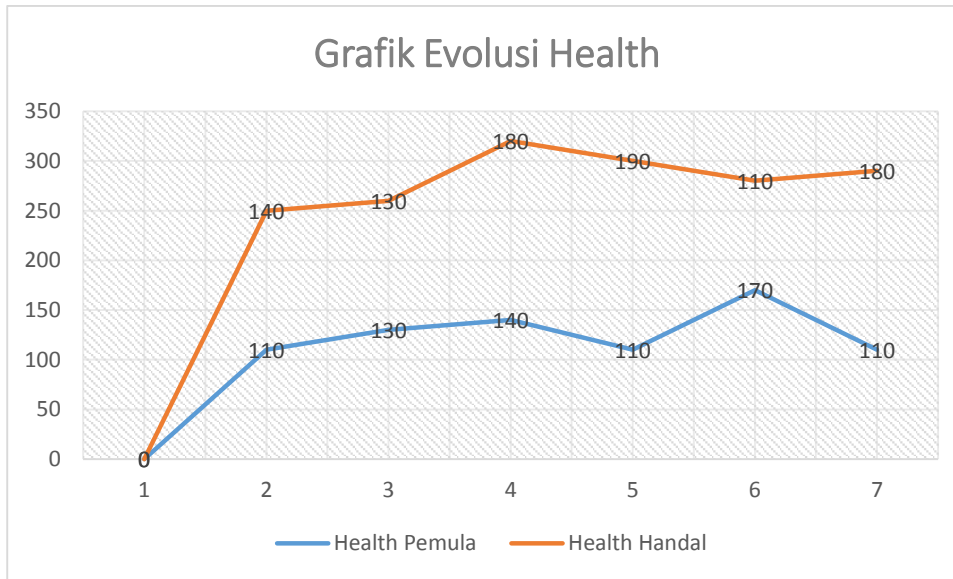
4.2.3 Grafik Perbandingan Evolusi Dinamis

Pada grafik perbandingan evolusi dinamis pada *speed* NPC antara 2 karakteristik *player* menunjukkan bahwa grafik perubahan dari *player* handal menanjak drastis dari pada *player* pemula. Terdapat perbedaan drastis antara 2 *player* dikarenakan kemampuan *player* handal berada diatas kemampuan *player* pemula. Seperti yang ditunjukkan pada Gambar 4.8



Gambar 4.8 Grafik Perbandingan Evolusi *Speed* NPC

Begitu pula pada grafik perbandingan evolusi dinamis pada *health* NPC antara 2 karakteristik *player* menunjukkan bahwa grafik perubahan dari *player* handal menanjak drastis dari pada *player* pemula. Terdapat perbedaan drastis antara 2 *player* dikarenakan kemampuan *player* handal berada diatas kemampuan *player* pemula. Seperti yang ditunjukkan pada Gambar 4.9



Gambar 4.9 Grafik Perbandingan Evolusi *Health* NPC

4.3 Simulasi NSGA - II

Tujuan dari simulasi menggunakan NSGA II adalah untuk mengeksplorasi dan menentukan konfigurasi NSGA-II yang tepat untuk menyelesaikan permasalahan di dalam *game* dengan genre *first person shooter*. Pada penelitian ini test-bed yang digunakan sebagai analisa dan uji coba permasalahan adalah *game space shooter*. Penelitian ini dikhususkan pada evolusi dinamis NPC (*health* dan *speed* pada pesawat NPC) dengan melihat performa evolusi dinamis NPC dengan menggunakan NSGA II. Simulasi dilakukan dalam beberapa model skenario untuk melihat solusi pareto-optimal dari evolusi dinamis pada *speed* dan *health* dari NPC.

4.3.1 Pengujian Performa NSGA - II

Simulasi performa NSGA II dilakukan dengan menumbuhkan beberapa model simulasi dengan jumlah populasi secara berurutan dengan jumlah populasi 50 dan jumlah *generasi* adalah 100. Pada simulasi terdapat 4 variabel yang menjadi variabel keputusan dalam penelitian ini. Dimana variable tersebut merujuk pada *objective function* yang diambil dari perilaku *player* selama uji simulasi dalam *game space shooter*. Ke-empat variable tersebut dijadikan sebagai variabel keputusan untuk dapat mengevolusi dinamis *speed* (f_1) dan *health* (f_2) dari musuh (NPC). Keempat variable keputusan untuk dapat merubah *objective function* dari *speed* (f_1) dan *health* (f_2) NPC tersebut adalah *scor* dari *player*, *health* dari *player*,

NPCmissed yang tidak dapat dihancurkan *player*, dan total *damage player* yang digunakan *player* untuk menghancurkan pesawat musuh (NPC).

Dari ke-empat variable keputusan tersebut maka dapat diambil nilai *objective function* dari *player* untuk dapat mengevolusi dinamis NPC. Nilai *objective function* diambil sebagai nilai variable keputusan untuk membentuk kromosom pada perilaku evolusi dinamis NPC untuk level berikutnya. Sehingga *player* mendapatkan perlawanan yang seimbang dari NPC.

Namun *objective function* lainnya dari *player*, seperti tipe senjata yang digunakan *player*, item yang didapatkan maupun digunakan oleh *player* seperti item *shield*, item *health*, item *weapon*, dan *speed* dari *player*. Serta *objective function* dari NPC seperti jenis senjata yang sedang digunakan oleh NPC (*gun*, *missile*, *rudal*, dan *suicide*) dapat diasumsikan sebagai variasi dari *player* dan NPC, sehingga tidak memberikan pengaruh kepada keputusan NPC yang dibangkitkan.

4.3.1.1 Pengujian NSGA II Pada Evolusi dinamis NPC

Berikut hasil percobaan NSGA – II untuk multi-objective pada evolusi dinamis NPC untuk meningkatkan *objective function speed* (f_1) dan *health* (f_2) dari NPC. Parameter algoritma genetika yang digunakan dalam pengujian ini adalah konfigurasi NSGA-II yang digunakan berdasarkan parameter yang digunakan Deb [29] yaitu probabilitas persilangan (p_c) = 1 dan probabilitas mutasi (p_m) = $1/n$, dimana n merupakan jumlah variabel keputusan dan variabel keputusan di dalam simulasi berjumlah 2 variabel. Index distribusi persilangan (η_c) = 20 dan mutasi (η_m)=20. Nilai dari tiap parameter untuk simulasi ditunjukkan pada Tabel 4.5

Tabel 4.5 Konfigurasi Simulasi NSGA-II

NO	Keterangan	Nilai
1	Ukuran populasi	50
2	Ukuran generasi	100
3	Probabilitas <i>crossover</i> (p_c)	1
4	indeks distribusi <i>crossover</i> (n_c)	20
5	Probabilitas mutasi (p_m)	$1/n$
6	indeks distribusi mutasi (n_m)	20

Hasil ujicoba NSGA II pada evolusi dinamis NPC telah dapat menemukan solusi *pareto-optimal* yang cukup baik pada simulasi evolusi dinamis NPC. Dimana solusi optimal yang diperoleh telah memenuhi *pareto-optimal* pada ujicoba dengan 50 populasi dan 100 *generasi*. *Pareto set* telah menunjukkan tingkat perubahan *pareto-optimal front* yang merupakan sekumpulan solusi dalam pengujian fungsi *objective* untuk mendapatkan nilai *fitness* yang sesuai dengan zona *flow*.

Dengan menggunakan *mutation* adaptif pada evolusi dinamis NPC, maka dapat dilakukan perbaikan nilai *fitness* rata-rata yang signifikan dibanding generasi sebelumnya untuk mendapatkan beberapa solusi yang optimal (*multi-objective solution*). Pada gambar 4.10 menunjukkan grafik penyebaran populasi yang menunjukkan *pareto-set* berubah pada generasi pertama hingga mencapai *pareto-optimal* pada *speed*(f_1) dan *health*(f_2).

Nilai fungsi obyektif berada pada *linear hyper-plane*: $\sum_{i=1}^n x_i \in f_j^m = 1$, di mana i adalah indeks *objective function* dari variabel keputusan dan n adalah jumlah *objective function* dari variabel keputusan. Sedangkan j adalah index *objective function* dari variabel evolusi dinamis NPC dan m adalah jumlah *objective function* dari variabel evolusi dinamis NPC, dimana jumlah variabel evolusi dinamis NPC dapat disymbolkan sebagai $m = 2$ (*speed* dan *health*).

Kesulitan dalam permasalahan ini adalah membentuk konvergensi pada *hyper-plane*. Ruang pencarian (search space) berjumlah 50 *pareto-optimal front* lokal. NSGA-II dengan ukuran populasi 50 dijalankan dalam 100 generasi menggunakan operator persilangan SBX ($\eta_c = 20$) dan operator mutasi polinomial ($\eta_m = 20$). Probabilitas persilangan adalah 1 dan probabilitas mutasi adalah $1/n$ digunakan pada simulasi ini.

Data uji yang digunakan dalam simulasi adalah data uji yang diambil dari perilaku *player* dalam satu level permainan. Untuk melihat performa NSGA II dalam mensimulasikan evolusi dinamis NPC. Data uji dapat dilihat pada tabel 4.6.

Tabel 4.6 Menunjukkan Data Yang Digunakan Dalam Simulasi NSGA II

No	Nama Data	Nilai
1	Total NPC Hancur	10
2	<i>Heathl Player</i>	100
3	<i>Damage NPC</i>	20

4	Total <i>Attack</i> NPC	200
5	Total <i>Spawn</i> NPC	30
6	<i>Damage</i> Player	50
7	Total <i>Attack</i> Player	300
8	<i>Speed</i>	10
9	<i>Speed</i> Evolution	0.9
10	<i>Health</i> Evolution	0.3

Pada pengujian, untuk melihat evolusi dinamis *health* dan *speed* dari NPC dengan nilai fungsi objective yang berbeda. Berikut Tabel Pengujiannya dengan dua objective yang berbeda (*health* dan *speed*) menggunakan NSGA-II:

Tabel 4.7 Tabel Evolusi Dinamis NPC Dengan 50 Populasi dan 100 Generasi

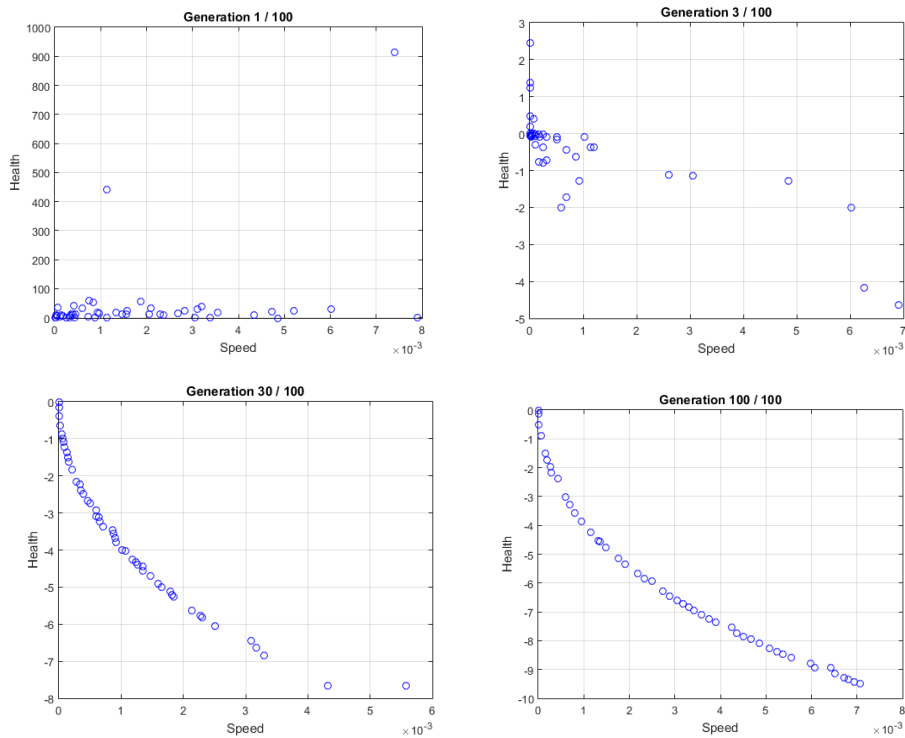
Data Ke	GenScore	GenHealth	GenNPC missed	Gen Damage	Speed NPC	Health NPC
91	0.486931	0.830226	1	1	0.507952	-8.26217
92	0.474437	0.680488	0.999773	1	0.359138	-7.10856
93	0.490781	0.418875	1	1	0.147988	-4.75336
94	0.490102	0.984112	1	1	0.681015	-9.35261
95	0.456657	0.397279	1	1	0.135779	-4.56879
96	0.461111	0.365211	1	1	0.115949	-4.24659
97	0.469212	0.00960289	1	1	1.225949	-0.127566
98	0.491706	0.874949	1	1	0.555744	-8.58138
99	0.432464	0.903871	1	1	0.598409	-8.80408
100	0.476307	0.393636	1	1	0.132567	-4.52233

Dari data uji pada tabel 4.4 diperoleh data hasil simulasi menggunakan NSGA II seperti yang terlihat pada tabel 4.5. Dari keseluruhan data yang diambil dari simulasi dengan 50 populasi dan 100 generasi, didapatkan data dengan peningkatan gen terfokuskan pada evolusi dinamis *speed* dari NPC dikarenakan *objective function speed* digunakan untuk meningkatkan parameter *speed* NPC. Sedangkan pada *objective function health* mengalami penurunan gen pada evolusi dinamis *health* dari NPC dikarenakan *objective function health* digunakan untuk menurunkan parameter *health* NPC.

Maka muncul sebuah paradigma, dimana level berikutnya pada permainan *space shooter*, *speed* NPC mengalami evolusi dinamis yang cukup tinggi (*speed*

dari NPC bertambah) sedangkan *health* NPC mengalami evolusi dinamis yang menurun (*health* dari NPC berkurang). Maka NPC dapat menyesuaikan diri terhadap perilaku *player* selama permainan berlangsung. Sehingga *player* dapat menjumpai perilaku NPC yang dinamis karena NPC dapat berevolusi sesuai dengan kemampuan *player*.

Seluruh data pada tabel 4.7 dapat membentuk sebuah *pareto-front* antara dua *objective function* yang berbeda atau saling bertentangan. Dimana grafik *pareto-front* yang biasa juga disebut *pareto-optimal* adalah sekumpulan solusi yang paling optimal dari setiap individu yang telah dibangkitkan setelah melewati proses evolusi genetic. Grafik *pareto-optimal* ditunjukkan pada Gambar 4.8.



Gambar 4.10. Experiment dengan 50 populasi dan 100 Generasi

Pada gambar 4.8 terlihat divergensi populasi pada simulasi generasi pertama dan selanjutnya pada generasi ke 30 terlihat populasi telah menemukan konvergensi penyebaran *objective function* dari NPC. Terlihat *objective function speed* (f_1) mengalami peningkatan populasi yang cukup tinggi dibandingkan dengan *objective function health* (f_2). Dikarenakan nilai *objective function* pada variabel keputusan *score player* cukup tinggi, nilai *objective function* pada variabel keputusan

health player cenderung rendah, *objective function* pada variabel keputusan *NPCmiss* cukup tinggi, dan *objective function* pada variabel keputusan *damage player* juga cukup tinggi.

Proses optimasi berlangsung untuk mendapatkan nilai fungsi *fitness* yang sesuai dengan kemampuan *player* sehingga dapat menemukan optimalisasi *objective function* dari NPC yang dapat menghadapi perilaku *player*.

Tujuan memeriksa setiap generasi ini adalah menentukan banyaknya generasi yang dibutuhkan di dalam permainan sehingga waktu untuk komputasi generasi berikutnya tidak membutuhkan waktu yang lama. Setiap individu yang ada pada front pertama diasumsikan sebagai solusi-solusi terbaik. Pengambil keputusan berhak memilih salah satu individu terbaik untuk digunakan pada permasalahannya. Dalam kasus *game space shooter* ini, solusi terbaik dipilih secara acak untuk mempercepat waktu komputasi saat *player* berhasil melanjutkan kelevel berikutnya atau bahkan harus kembali kelevel sebelumnya.

Halaman ini sengaja dikosongkan

BAB V

KESIMPULAN

5.1 Kesimpulan

NSGA-II dapat digunakan untuk memilih evolusi dinamis pada *speed* dan *health* dari NPC secara otomatis. Metode ini berhasil memilih evolusi yang adaptif sesuai dengan kondisi *player* yang sedang dihadapi oleh NPC. Simulasi yang dilakukan pada skenario dari permainan *space shooter* dapat membuktikan bahwa NSGA-II memberikan hasil yang optimal di setiap skenario tersebut.

NSGA II dapat memberikan solusi *multi-objectif* sesuai dengan perilaku dari *player*. Dimana dalam simulasi terdapat perubahan *speed* yang mulai naik dan jumlah *health* yang mulai berkurang. Dikarenakan pada simulasi dengan menggunakan NSGA II, parameter dari *player* yang sedang dihadapi membutuhkan evolusi pada *speed* agar dapat menghindari serangan *player*. Namun dikarenakan *speed* yang tinggi, maka *health* harus lebih rendah agar dapat menyesuaikan dengan kemampuan *player*.

NSGA-II mampu mengoptimalkan pemilihan evolusi dinamis *speed* dan *health* dari NPC dengan melihat beberapa parameter dari *player*. Keempat parameter utama yang digunakan untuk evolusi dinamis *speed* dan *health* dari NPC antara lain *score player*, *health player*, *NPCMissed*, dan *damage player* yang digunakan untuk menghancurkan pesawat musuh yang dikendalikan oleh *non-player character* (NPC).

5.2 Saran

Evolusi dinamis *speed* dan *health* NPC dalam *game space shooter* ini merupakan salah satu topik riset dalam pengembangan *artificial intelligence* pada pemilihan *difficulty level*. Beberapa saran untuk pengembangan sistem terutama berhubungan dengan DDA dan NSGA-II, yaitu:

1. Melakukan percobaan dengan model multi-obyektif lainnya sehingga bisa membandingkan matriks performa terhadap permasalahan yang dihadapi.

2. Melakukan percobaan dengan menggabungkan semua elemen dari permainan dengan *genre first person shooter* untuk topik penelitian *Procedural Content Generation*.

DAFTAR PUSTAKA

- [1] Camerer, Colin, Teck Ho, and Kuan Chong. "Models of thinking, learning, and teaching in games." *American Economic Review* (2003): 192-195.
- [2] Haryono, Alm, and Ari Utomo. "Optimalisasi Penerimaan Pajak Hiburan Dalam Rangka Meningkatkan Pendapatan Asli Daerah (Studi Pada Pemerintah Kota Bandar Lampung Tahun 2011)." (2013).
- [3] Muhammad, Aidi. "Analisis Optimalisasi Pelayanan Konsumen Berdasarkan Teori Antrian Pada Kaltimgps. Com Di Samarinda." (2014).
- [4] Laksma Paramestha, Dominico. *Penerapan Multi-Objective Particle Swarm Optimization Untuk Kasus Capacitated Vehicle Routing Problem Dengan Load Balancing*. Diss. Uajy, 2014.
- [5] Naplayerra, J., & Csikszentmihalyi, M., (2002), "The concept of flow". *Handbook of positive psychology*, hal. 89 -105
- [6] Chen, J., (2007), "Flow in Games (and Everything Else)", *Communications of the ACM Viewpoint Paper*, hal 31 – 34
- [7] Ijsselsteijn W., de Kort Y., Poels K., Jurgelionis A., Bellotti F., (2007), "Characterizing and Measuring User Experiences in Digital Games", *International Conference on Advances in Computer Entertainment Technology*, Vol. 2, hal. 27 – 28
- [8] Jacob Schrum dan Risto Mikkulainen, "Evolving Multimodal Networks for Multitask Games", (2012), *IEEE Transactions on Computational Intelligence and AI in Games*, Vol. 4, No. 2, hal. 94 – 111.
- [9] Kenneth O. Stanley dan Risto Mikkulainen, (2002), "Evolving Neural Networks Through Augmenting Topologies". *Evolutionary Computation*, Vol. 10, No. 2, hal. 99 – 127.
- [10] Ariyadi, *Game Technology ITS* (2012), "Evolusi Penyerangan NPC Untuk Pengaturan Tingkat Kesulitan Dinamis Pada *Game Tower Defense* Dengan Metode NSGA-II"
- [11] Rahmat Fauzi, *Game Technology ITS* (2013), "Perilaku NPC Saat Pertahanan Tempur Menggunakan *Hierarchical Finite State Machine* (HFSM)"

- [12] Widowati Anantasari (2015), Efektifitas Penggunaan Media Video Berbasis Lingkungan (Mvbl) Dalam meningkatkan Perilaku Positif Anak Terhadap Lingkungan Optimasi, Jurnal Penelitian Pendidikan IPA (JPPIPA)
- [13] Deb K. (1995), *Optimization for engineering design: algorithms and examples*, Prentice-Hall, New Delhi, India.
- [14] Deb K. (2001), *Multi-objective Optimization using Evolutionary Algorithms*. Wiley and Sons, Chichester, UK.
- [15] Deb, Kalyanmoy, et al. "A fast and elitist multiobjective genetic algorithm: NSGA-II." *Evolutionary Computation, IEEE Transactions on* 6.2 (2002): 182-197.
- [16] David Goldberg, "The Design of Innovation (Genetic Algorithms and Evolutionary Computation)", Springer; 1 edition (June 30, 2002).
- [17] Tobing, Reynold Lumban. "Sistem Simulasi Penjadwalan Kuliah Dengan Menggunakan Algoritma Genetik." (2010).
- [18] Kuku Adisusilo, Anang. "Optimasi Perilaku Agen Pada Game Tinju Menggunakan Algoritma Genetika" *Optimasi Perilaku Agen Pada Game Tinju Menggunakan Algoritma Genetika* (2013).
- [19] Srinivas, Nidamarthi, and Kalyanmoy Deb. "Multiobjective optimization using nondominated sorting in genetic algorithms." *Evolutionary computation* 2.3 (1994): 221-248.
- [20] Deb K. (1995), *Optimization for engineering design: algorithms and examples*, Prentice-Hall, New Delhi, India.
- [21] Kuang, A., & Lextraire, T., (2012), "Dynamic Difficulty Adjustment", *Worcester Polytechnic Institute Final Report*, Massachusetts, US.
- [22] Deb K, Thiele L, Laumanns M, dan Zitzler E, (2002), "Scalable Multi-Objective Optimization Test Problems", *Congress on Evolutionary Computation 2002*, Vol 1, hal. 825-830
- [23] Deb K, Agarwal RB, (1995), "Simulated Binary Crossover for Continuous Search Space", *Complex Systems*, Vol. 9, hal 115-148.
- [24] Raghuwanshi MM, Kakde OG (2004) "Survey on Multiobjective Evolutionary and Real Coded Genetic Algorithms", *Proceedings of the 8th*

- Asia Pacific Symposium on Intelligent and Evolutionary Systems*, hal. 150-161.
- [25] Sweetser, P., & Wiles, J., (2005), "Scripting Versus Emergence: Issues for Game Developers and Players in Game Environment Design". *International Journal of Intelligent Games and Simulations*, Vol. 4, hal. 1-9.
 - [26] Hunicke, R., & Chapman, V., (2004), "AI for Dynamic Difficulty Adjustment in Games, *Challenges in Game Artificial Intelligence AAAI Workshop*, hal. 91-96.
 - [27] Missura, O., & Gärtner, T., (2009), "Player Modelling for Intelligent Difficulty Adjustment". *Discovery Science*, hal. 197-211.
 - [28] Holland, J.H. (1975), "Adaptation in natural and artificial systems". *Ann Arbor, MI, USA: University of Michigan Press*.
 - [29] Deb K, Agrawal S, Pratap A, Meyarivan T. (2002), "A fast and elitist multi-objective genetic algorithm: NSGA-II", *IEEE Transactions on Evolutionary Computation*, Vol. 6 No. 2, hal. 182–197.
 - [30] Chakraborty UK. (2008), "Advances in differential evolution. Studies in computational intelligence", vol. 143. Heidelberg, Germany: Springer;
 - [31] Deb K, Thiele L, Laumanns M, dan Zitzler E, (2002), "Scalable Multi-Objective Optimization Test Problems", *Congress on Evolutionary Computation 2002*, Vol 1, hal. 825-830
 - [32] Tremblay, J., Bouchard, B., & Bouzouane, A., (2010), "Adaptive Game Mechanics for Learning Purposes-Making Serious Games Playable and Fun", *International Conference on Computer Supported Education*, Vol. 2, hal. 465 – 470.
 - [33] Liu, C., Agrawal, P., Sarkar, N., & Chen, S., (2009), "Dynamic Difficulty Adjustment in Computer Games through Real-time Anxiety-based Affective Feedback". *International Journal of Human–Computer Interaction*, Vol. 25, hal. 506 - 529.
 - [34] Yidan, Z., H. Suoju, et al., (2010), "Optimizing Player's Satisfaction through DDA of Game AI by UCT for the Game Dead-End". *2010 Sixth International Conference on Natural Computation*.

Halaman ini sengaja dikosongkan

BIODATA DIRI



DATA PRIBADI

Nama : Darmawan Aditama
Tempat, tanggal lahir : Lamongan, 16 November 1990
Alamat Rumah : Jln. Tlogo Ploso Payaman - Solokuro - Lamongan
Jenis Kelamin : Laki-laki
Agama : Islam
Telepon / HP : 085648549079
Email : awantamahone@gmail.com
Hobby : Membaca, Diskusi, Mancing, Dan Olah Raga

RIWAYAT PENDIDIKAN

MI Muhammadiyah 05 Palirangan	1997 - 2003
MTS Muhammadiyah 12 Palirangan	2003 - 2006
MA Al-Ishlah Sendang Agung	2006 - 2009
Universitas Trunojoyo Madura	2009 - 2014
Institut Teknologi Sepuluh Noverber	2014 - 2017

